À la recherche du détecteur de fautes minimal pour le k-accord

Julien Stainer

UNIVERSITÉ DE RENNES 1

Rapport de stage
Master recherche en informatique

À la recherche du détecteur de fautes minimal pour le k-accord

Auteur :
Julien STAINER

Encadrant :
Michel RAYNAL [1]

1. Institut Universitaire de France et Projet ASAP (équipe commune avec l'INRIA, le CNRS, l'université de Rennes 1 et l'INSA de Rennes), raynal@irisa.fr

Résumé

Ce rapport de stage de master recherche en informatique présente des travaux effectués dans le domaine des systèmes distribués asynchrones. Il s'agit d'études visant à déterminer le plus faible détecteur de faute permettant de résoudre le problème du k-accord dans un système distribué asynchrone où les processus communiquent par passage de messages et où un nombre arbitrairement grand de défaillances peuvent survenir.

Plusieurs relations entre les détecteurs de fautes existants sont mises en évidence, en particulier, ce qui sépare $\mathcal{L}_k$ de Σ_k ou de Ω_k. Deux nouveaux détecteurs de fautes suffisants pour résoudre le k-accord, $\Box\Pi_k$ et $\Pi\Sigma_{x,y}$ (avec $xy \leq k$), sont introduits. Nous prouvons que $\Box\Pi_k$ est, hormis dans les cas extrêmes, strictement plus faible que $\mathcal{L}_k$ et que $\mathcal{S}_{n-k+1}$ (qui sont deux détecteurs de fautes également suffisants mais incomparables). Nous démontrons également que, lorsque $1 \leq y < x \leq n-1$, le détecteur de fautes $\Pi\Sigma_{x,y}$ est strictement plus faible que la paire de détecteurs $\langle \Sigma_x, \overline{\Omega}_y \rangle$ (qui a été étudiée pour la résolution du xy-accord).

Par ailleurs, nous nous appuyons sur $\Pi\Sigma_{x,y}$ pour démontrer que Ω_k, bien qu'il soit plus faible que plusieurs des détecteurs proposés pour résoudre le k-accord, n'est pas nécessaire à cette résolution.

Table des matières

Chapitre 1

Introduction

Ce rapport a pour objectif de présenter les résultats obtenus lors de mon stage de master recherche en informatique effectué au sein de l'équipe IRISA/INRIA ASAP. Les travaux présentés dans la suite ont été réalisés en collaboration avec Achour MOSTÉFAOUI et avec Michel RAYNAL qui encadrait ce stage.

Le problème abordé ici est celui de la recherche des hypothèses minimales pour pouvoir résoudre le problème du k-accord [8] dans un système distribué. Résoudre le k-accord consiste à fournir aux processus un objet distribué qui leur permet de proposer une valeur et qui leur fournit en retour une valeur à décider. Toute valeur décidée doit être une des valeurs proposées et, dans tout le système, au plus k valeurs différentes sont décidées. Ce problème est une généralisation du problème, fondamental dans les systèmes distribués, du consensus (1-accord).

Il a été montré [4, 15, 24] que, dans un système distribué asynchrone sujet à un nombre quelconque de défaillances, le problème du k-accord était impossible à résoudre. Il est alors naturel de se demander sous quelles hypothèses cette impossibilité est levée. Pour modéliser précisément ces hypothèses, les détecteurs de fautes (ou oracles distribués) ont été introduits [6]. Il s'agit de fournir aux processus du système des informations sur les défaillances. Le problème se reformule alors sous cette forme : quelle est l'information minimale sur les défaillances qui permet aux processus de résoudre le k-accord.

Ce problème a déjà été étudié, et plusieurs détecteurs de fautes ont été proposés pour le résoudre. Lorsque les processus communiquent à l'aide d'une mémoire partagée, le détecteur de fautes minimal (celui qui apporte l'information nécessaire et suffisante) pour résoudre le k-accord est connu quelle que soit la valeur de k. Dans le modèle plus général de la communication par passage de messages, les détecteurs de fautes minimaux pour le 1-accord et pour le $n-1$-accord (où n est le nombre de processus) sont connus, mais, dans les cas où $1 < k < n-1$ on ne les a pas encore découverts. L'objectif de ce stage a été de progresser dans leur recherche.

Le chapitre 2 de ce rapport présente le modèle de système distribué que nous étudions, définit le problème du k-accord et présente les détecteurs de fautes adaptés à ce problème et connus au début de ce stage. Au cours du chapitre 3, plusieurs de ces détecteurs sont mis en relation et étudiés. Ensuite, le chapitre 4 introduit un détecteur de faute, suffisant pour le k-accord, et qui apporte moins d'information que deux parmi ceux que nous connaissions. Enfin, un autre détecteur de faute sera présenté dans le chapitre 5, il affaiblit un autre des détecteurs utilisé pour le k-accord et nous permet de montrer que certaines hypothèses (représentées par un détecteur de faute) ne sont pas nécessaires à la résolution du k-accord, bien qu'elles aient été fréquemment utilisées jusque là.

Chapitre 2

Modèles, problématique et état de l'art

2.1 Modèle sous-jacent et notations

2.1.1 Système distribué asynchrone

Nous considérons un système distribué constitué d'un ensemble de n processus. Chaque processus est muni d'un identifiant unique ; nous considèrerons que ces identifiants sont choisis dans l'intervalle d'entiers $\mathcal{P} = [\![1; n]\!]$. Nous noterons p_i le processus d'identité $i \in \mathcal{P}$. Pour éviter de surcharger les preuves et éviter les formulations du type « les processus dont les identifiants apparaissent dans S », nous assimilerons parfois un ensemble de processus et l'ensemble de leurs identités.

Les processus exécutent une suite d'intructions élémentaires qui peuvent être des calculs locaux ou des appels à des primitives de communication. Aucune hypothèse de synchronie n'est faite sur les vitesses relatives des processus.

2.1.2 Défaillances

Le modèle de défaillances étudié ici est celui du *crash*. Un processus peut à tout moment cesser de fonctionner, il n'exécute alors plus d'instructions. Pour une exécution donnée, nous noterons $\mathcal{F}$ l'ensemble des identités de processus *fautifs* (ceux qui subissent une défaillance à un moment de l'exécution), et $\mathcal{C} = \mathcal{P} \setminus \mathcal{F}$ l'ensemble des identités de processus *corrects*.

Nous noterons t le nombre maximum de processus qui peuvent être défaillants lors d'une exécution. Sauf indication contraire, nous nous placerons toujours dans le cas où $t = n - 1$. Par conséquent, $1 \leq |\mathcal{C}| \leq n$.

2.1.3 Communication par passage de messages

Dans le cadre de notre étude, la communication inter-processus s'appuie sur un réseau permettant l'envoi et la réception de messages. Nous supposerons que ce réseau est complètement maillé : il y a un canal bidirectionnel entre chaque paire de processus.

Les canaux sont asynchrones : on sait que le temps mis par un message pour parvenir au destinataire est fini, cependant il n'est pas borné *a priori*. Les canaux seront supposés fiables : ils ne perdent pas, n'altèrent pas, ne dupliquent pas, ni ne créent de messages. Le modèle de

système distribué asynchrone communicant par passage de messages sera noté $\mathcal{AMP}$ (*Asynchronous Message-Passing*) dans la suite.

On utilisera broadcast T_MSG(arg) comme raccourci pour : **for each** $j \in \mathcal{P}$ **do** send T_MSG(arg) to p_j **endfor**. Une diffusion n'est pas atomique, un processus peut cesser de fonctionner au milieu d'une diffusion et par conséquent ne pas envoyer le message à certains processus.

Nous évoquerons parfois le modèle de communication par mémoire partagée. Dans ce modèle, les processus partagent une mémoire à laquelle ils accèdent à l'aide de primitives de lecture/écriture. Il sera noté $\mathcal{ASM}$ (*Asynchronous Shared Memory*).

2.1.4 Modèlisation du temps

Conformément aux hypothèses d'asynchronie, aucun temps global n'est disponible sur les processus. Cependant, nous considèrerons un ensemble d'instants pris dans l'ensemble des entiers naturels afin de décrire l'ordonancement des évènements du point de vue d'un observateur extérieur. Cette notion de temps sera utilisée comme un outil pour énoncer ou prouver certaines propriétés.

2.2 Problème du k-accord

2.2.1 Définition

Le problème du k-accord, introduit par S. CHAUDHURI [8], est un problème de décision : chaque processus propose une valeur puis décide une valeur (s'il ne crashe pas). Les propriétés suivantes doivent être vérifiées :

Validité : Les valeurs décidées doivent avoir été proposées.

Terminaison : Tout processus correct décide en un temps fini.

Accord : Au plus k valeurs différentes sont décidées dans l'ensemble du système.

Ce problème peut être résolu de façon très simple dans le cas où $t < k$ (lorsque le nombre de défaillances ne peut pas égaler ou excéder le nombre maximal de valeurs qui peuvent être décidées). Il suffit alors que k processus choisis *a priori* (les k de plus petites identités par exemple) diffusent la valeur qu'ils proposent et qu'un processus décide la première valeur qu'il reçoit. Par conséquent, le problème du k-accord n'est intéressant que dans le cas où $t \geq k$.

Il a été montré dans [4, 15, 24] que dans la configuration $t \geq k$ le problème du k-accord est impossible à résoudre dans $\mathcal{ASM}$ (le modèle $\mathcal{ASM}$ permettant de simuler $\mathcal{AMP}$, l'impossibilité est également valide dans ce dernier).

2.2.2 Cas particuliers du consensus et de l'accord ensembliste

Le problème du k-accord a tout d'abord été introduit et étudié dans la configuration $k = 1$, on le nomme alors *consensus*. Ce problème est fondamental dans le domaine de l'algorithmique distribué, sa résolution permet notamment de répliquer des machines à états afin d'assurer un service résistant aux pannes.

À l'autre extrémité du spectre (mis à part le cas $k = n$ dans lequel il suffit que chaque processus décide la valeur qu'il propose), le $n-1$-accord est appelé *accord ensembliste*. Il s'agit de la plus faible version non triviale du k-accord.

2.3 Les détecteurs de fautes

2.3.1 Principe

Dans les systèmes asynchrones, l'impossibilité de résoudre certains problèmes est généralement directement liée au fait qu'il est impossible pour un processus de savoir si un autre processus est défaillant ou s'il est juste lent. La question qui se pose alors est de savoir quelle information sur les défaillances doit être fournie aux processus afin de résoudre un problème donné. Pour répondre à cette question, les détecteurs de fautes (ou oracles distribués) ont été introduit dans [6]. Il s'agit d'équiper les processus avec des variables accessibles en lecture seule dont les valeurs les informent sur les défaillances qui ont lieu au cours de l'exécution.

2.3.2 Notations

Si un détecteur de fautes $\mathcal{D}$ fournit aux processus une variable d, nous noterons $\mathcal{AMP}[\mathcal{D}]$ le système obtenu en enrichissant le modèle $\mathcal{AMP}$ avec le détecteur de fautes $\mathcal{D}$ et nous désignerons par d_i^τ la valeur de la variable d du processus p_i à l'instant τ.

2.3.3 Comparaisons

Pour un modèle $\mathcal{M}$ donné, on introduit un ordre partiel sur l'ensemble des détecteurs de fautes. Soient $\mathcal{D}$ et $\mathcal{D}'$ deux détecteurs de fautes. Si il existe un algorithme qui simule $\mathcal{D}'$ dans $\mathcal{M}[\mathcal{D}]$, on dit alors que $\mathcal{D}$ est plus fort que $\mathcal{D}'$ (ou que $\mathcal{D}'$ est plus faible que $\mathcal{D}$) et on note $\mathcal{D} \succeq_{\mathcal{M}} \mathcal{D}'$ (ou $\mathcal{D}' \preceq_{\mathcal{M}} \mathcal{D}$). Dans les cas où il n'y a pas d'ambiguïté, on omettra de préciser le modèle sous-jacent.

Si $\mathcal{D} \succeq_{\mathcal{M}} \mathcal{D}'$ et $\mathcal{D} \preceq_{\mathcal{M}} \mathcal{D}'$, on dit alors que $\mathcal{D}$ et $\mathcal{D}'$ sont équivalents et on note $\mathcal{D} \simeq_{\mathcal{M}} \mathcal{D}'$. Si $\mathcal{D} \succeq_{\mathcal{M}} \mathcal{D}'$ et $\mathcal{D} \not\preceq_{\mathcal{M}} \mathcal{D}'$, on dit que $\mathcal{D}$ est strictement plus fort que $\mathcal{D}'$ (ou que $\mathcal{D}'$ est strictement plus faible que $\mathcal{D}$) et on note $\mathcal{D} \succ_{\mathcal{M}} \mathcal{D}'$ (ou $\mathcal{D}' \prec_{\mathcal{M}} \mathcal{D}$). Enfin, si $\mathcal{D} \not\succeq_{\mathcal{M}} \mathcal{D}'$ et $\mathcal{D} \not\preceq_{\mathcal{M}} \mathcal{D}'$, on dit alors que $\mathcal{D}$ et $\mathcal{D}'$ sont incomparables.

2.3.4 Caractérisation des problèmes par le détecteur de fautes minimal associé

Soient un modèle $\mathcal{M}$ et un problème P. On dit d'un détecteur de fautes $\mathcal{D}_{min}$ qu'il est minimal pour résoudre P dans $\mathcal{M}$ s'il existe un algorithme qui résoud P dans $\mathcal{M}[\mathcal{D}_{min}]$ et si, pour tout détecteur de fautes $\mathcal{D}$ tel qu'il existe un algorithme qui résoud P dans $\mathcal{M}[\mathcal{D}]$, on a $\mathcal{D}_{min} \preceq \mathcal{D}$.

Il est montré dans [16] qu'un tel détecteur de faute existe. Si deux détecteurs de fautes incomparables permettent de résoudre le problème, on peut en définir un troisième, plus faible que les deux précédents, et qui permet toujours de résoudre le problème.

Ce détecteur minimal permet de caractériser les informations minimales sur les défaillances qui doivent être fournies aux processus pour résoudre P dans $\mathcal{M}$. L'ordre sur les détecteur de fautes induit alors naturellement une hiérarchie sur l'ensemble des problèmes dans un modèle donné.

2.4 Détecteurs existants et résultats correspondants

Depuis qu'il a été énoncé, le problème du k-accord a entrainé de nombreuses études visant à déterminer le détecteur de fautes minimal pour le résoudre. Si dans $\mathcal{ASM}$ ce détecteur à été découvert, ce n'est pas le cas dans $\mathcal{AMP}$.

Cette section présente des détecteurs de fautes qui existaient au début de notre étude et qui s'approchent du détecteur minimal pour le k-accord dans $\mathcal{AMP}$.

2.4.1 Σ_k

Σ_k, introduit dans [3], est une généralisation de Σ ($\Sigma_1 \simeq \Sigma$). Σ a été introduit dans [10] et il est prouvé [10, 2] qu'il s'agit du détecteur de faute minimal pour implémenter une mémoire partagée dans $\mathcal{AMP}$. Intuitivement, Σ_k empêche l'ensemble des processus d'être partionné en plus de k parties.

Formellement, Σ_k équipe les processus d'une variable qr contenant un ensemble d'identités de processus appelé *quorum*. Les propriétés suivantes doivent être vérifiées :

Vivacité : $\exists \tau : \forall i \in \mathcal{C}, \forall \tau' \geq \tau : qr_i^{\tau'} \subseteq \mathcal{C}$

Intersection : $\forall (id_m)_{1 \leq m \leq k+1} \in \mathcal{P}^{k+1}, \forall (\tau_m)_{1 \leq m \leq k+1} \in \mathbb{N}^{k+1}$:

$$\exists (i,j) \in [\![1; k+1]\!]^2 : (i \neq j) \wedge (qr_{id_i}^{\tau_i} \cap qr_{id_j}^{\tau_j} \neq \emptyset)$$

La propriété de vivacité assure qu'inéluctablement les quorums des processus corrects ne contiennent que des identités de processus corrects. La propriété d'intersection garantit qu'un ensemble de $k+1$ quorums pris à des instants quelconques sur n'importe quels processus en contient toujours deux qui s'intersectent.

Il a été montré dans [3] que, quelque soit k, Σ_k est nécessaire pour résoudre le k-accord dans $\mathcal{AMP}$. C'est à dire que, pour tout détecteur de fautes $\mathcal{D}$, s'il existe un algorithme qui résoud le k-accord dans $\mathcal{AMP}[\mathcal{D}]$, alors on peut simuler Σ_k dans $\mathcal{AMP}[\mathcal{D}]$. Autrement dit, Σ_k est plus faible que le détecteur de fautes minimal pour résoudre le k-accord dans $\mathcal{AMP}$.

Par ailleurs, il est également montré dans [3] que Σ_{n-1} permet de résoudre le $n-1$-accord. Il correspond donc au détecteur minimal pour le $n-1$-accord dans $\mathcal{AMP}$.

Un algorithme résolvant le k-accord dans $\mathcal{AMP}[\Sigma_z]$ pour $k \geq n - \lfloor \frac{n}{z+1} \rfloor$ a également été présenté dans [5] et il y est montré que lorsque cette condition sur z et k n'est pas vérifiée il est impossible de résoudre le k-accord dans ce système.

2.4.2 Ω_k et $\overline{\Omega}_k$

Les détecteurs Ω_k et $\overline{\Omega}_k$ sont des généralisations d' Ω (introduit dans [7]) détecteur de fautes minimal pour résoudre le consensus dans $\mathcal{ASM}$ ($\Omega \simeq \Omega_1 \simeq \overline{\Omega}_1$).

Ω_k et $\overline{\Omega}_k$ fournissent à chaque processus une variable $leaders$ contenant un ensemble de k identités de processus. On définit les propriétés suivantes :

Validité : $\forall i \in \mathcal{P}, \forall \tau : (leaders_i^\tau \subseteq \mathcal{P}) \wedge (|leaders_i^\tau| = k)$

Autorité faible inéluctable : $\exists \ell \in \mathcal{C}, \exists \tau : \forall \tau' \geq \tau, \forall i \in \mathcal{C} : \ell \in leaders_i^{\tau'}$

Autorité forte inéluctable : $\exists LD \subseteq \mathcal{P} : (LD \cap \mathcal{C} \neq \emptyset) \wedge (\exists \tau : \forall \tau' \geq \tau, \forall i \in \mathcal{C} : leaders_i^{\tau'} = LD)$

Ω_k [20] est la famille des détecteurs de fautes qui vérifient les propriétés de validité et d'autorité forte inéluctable. Ce détecteur assure que les processus corrects finissent par s'accorder sur un même ensemble de k leaders dont au moins un est correct. $\overline{\Omega}_k$ [21, 25] est la famille de ceux qui vérifient les propriétés de validité et d'autorité faible inéluctable. Ce détecteur est plus faible (il suffit de prendre $\ell \in LD \cap \mathcal{C}$ pour constater que $\overline{\Omega}_k \preceq \Omega_k$), il assure uniquement qu'un même processus correct apparaitra inéluctablement dans tous les ensembles $leaders$ des processus corrects. On remarque que ces détecteurs autorisent une phase arbitrairement longue d'anarchie (pendant laquelle leurs sorties peuvent être quelconques).

Pour tout k, $\overline{\Omega}_k$ est le détecteur de faute minimal pour résoudre le k-accord dans $\mathcal{ASM}$ [13]. Par ailleurs, le détecteur de faute minimal pour résoudre le consensus dans $\mathcal{AMP}$ est celui constitué de la paire $\langle \Sigma, \Omega \rangle$ (il s'agit d'un détecteur qui fournit aux processus une paire de variables dont un des éléments vérifie les propriétés de Σ et l'autre celles de Ω) [10]. Cependant, dès que $k > 1$, la paire $\langle \Sigma, \overline{\Omega}_k \rangle$, si elle permet de résoudre le k-accord dans $\mathcal{AMP}$ (on simule une mémoire partagée avec Σ et l'on peut alors résoudre le k-accord avec $\overline{\Omega}_k$), n'est pas le détecteur de fautes minimal pour ce problème dans ce système [9].

2.4.3 Π_k

Π_k, introduit dans [3], est une version renforcée de Σ_k. Il fournit aux processus une variable qr qui, en plus de vérifier les propriétés de Σ_k, assure la propriété suivante :

Autorité inéluctable : $\exists LD \subseteq \mathcal{P} : (|LD| = k) \wedge (\exists \tau : \forall \tau' \geq \tau, \forall i \in \mathcal{C} : qr_i^{\tau'} \cap LD \neq \emptyset)$

Il est montré dans [3] que $\Pi_k \simeq \langle \Sigma_k, \Omega_k \rangle$. Π_k correspond aux détecteurs de fautes minimaux pour les cas $k = 1$ ($\Pi_1 \simeq \langle \Sigma, \Omega \rangle$) et $k = n - 1$ ($\Pi_{n-1} \simeq \Sigma_{n-1}$). Cependant, il est prouvé dans [5] que Π_k ne permet pas de résoudre le k-accord lorsque $1 < k < n - 1$ et $2k^2 \leq n$.

2.4.4 $\langle \Sigma_x \times \overline{\Omega}_y \rangle$

Le détecteur formé du couple $\langle \Sigma_x \times \overline{\Omega}_y \rangle$ a été étudié dans [5]. Un algorithme qui résoud le k-accord dans $\mathcal{AMP}[\langle \Sigma_x \times \overline{\Omega}_y \rangle]$ pour $k \geq xy$ est présenté dans cet article et il y est démontré qu'il n'existe pas dans ce système d'algorithme pour le k-accord lorsque $k < xy$ et $2xy \leq n$.

2.4.5 $\mathcal{S}_{n-k+1}$

Le détecteur $\mathcal{S}_x$ [18, 17] est une généralisation de $\mathcal{S}$ introduit dans [6] ($\mathcal{S} \simeq \mathcal{S}_n$). Il fournit aux processus une variable $suspected$ qui vérifie les propriétés suivantes :

Complétude : $\exists \tau : \forall \tau' \geq \tau, \forall i \in \mathcal{C} : \mathcal{F} \subseteq suspected_i^{\tau'}$

x-précision : $\exists S \subseteq \mathcal{P} : (|S| = x) \wedge (\exists \ell \in S \cap \mathcal{C} : \forall i \in S, \forall \tau : \ell \notin suspected_i^\tau)$

La propriété de complétude assure qu'inéluctablement, tous les processus fautifs seront suspectés par les processus corrects. La propriété de x-précision garantit l'existence d'un sous-ensemble de processus qui ne suspecteront jamais l'un d'entre eux.

Un algorithme qui résoud le k-accord dans $\mathcal{AMP}[\mathcal{S}_{n-k+1}]$ est présenté dans [17].

2.4.6 $\mathcal{L}_k$

Le détecteur de fautes $\mathcal{L}_k$ [1] est une généralisation de $\mathcal{L}$ [11] (qui correspond au détecteur de fautes minimal pour l'accord ensembliste $\mathcal{L} \simeq \mathcal{L}_{n-1} \simeq \Sigma_{n-1}$. $\mathcal{L}_k$ fournit aux processus une variable booléenne $alone$ qui vérifie les propriétés suivantes :

Vivacité : $|\mathcal{F}| \geq k \implies \exists \ell \in \mathcal{C}, \exists \tau : \forall \tau' \geq \tau : alone_\ell^{\tau'}$

Sûreté : $|\{i \in \mathcal{P} | \exists \tau : alone_i^\tau\}| \leq k$

La propriété de vivacité assure que dans les exécutions où plus de k processus sont défaillants, un des processus corrects se considèrera inéluctablement et pour toujours comme seul. La propriété de sûreté garantit qu'au plus k processus auront l'impression d'être seuls au court d'un exécution.

Un algorithme qui résoud le k-accord dans $\mathcal{AMP}[\mathcal{L}_k]$ a été présenté dans [1]. S'il est montré dans cet article que $\mathcal{L}_k$ n'est pas le détecteur minimal pour le k-accord dans $\mathcal{AMP}$ pour $k < n-1$ (en effet dans ces cas, $\mathcal{S}_{n-k+1}$ et $\mathcal{L}_k$ sont incomparables), les auteurs y prouvent que le k-accord est impossible à résoudre dans $\mathcal{AMP}[\mathcal{L}_{k+1}]$. $\mathcal{L}_k$ est donc le plus faible de sa famille permettant la résolution de notre problème.

2.4.7 Vue d'ensemble

Pour résumer la situation, nous avons à notre disposition plusieurs détecteurs de fautes suffisants pour résoudre le k-accord dans $\mathcal{AMP}$:

- Σ_z pour z tel que $k \geq n - \lfloor \frac{n}{z+1} \rfloor$
- $\langle \Sigma_x, \overline{\Omega}_y \rangle$ pour x et y tels que $k \geq xy$
- $\mathcal{S}_{n-k+1}$
- $\mathcal{L}_k$

Par ailleurs, nous savons que Σ_k est nécessaire pour résoudre le k-accord dans $\mathcal{AMP}$. Autrement dit, tout détecteur permettant cette résolution est plus fort que Σ_k.

Pour les cas extrêmes du consensus et de l'accord ensembliste, les détecteurs de fautes minimaux sont connus. Parmi les détecteurs de fautes présentés, certains coïncident avec le détecteur minimal pour $k = 1$: $\langle \Sigma_1, \overline{\Omega}_1 \rangle$ et Π_1. Ceux qui coïncident avec le détecteur optimal pour $k = n - 1$ sont Σ_{n-1}, $\mathcal{L}_{n-1}$ et Π_{n-1}.

Chapitre 3

Étude des détecteurs existants

Après le travail de bibliographie qui a précédé ce stage, nous avions à notre disposition un bestiaire de détecteurs de fautes plus ou moins bien adaptés à notre problème. L'objectif étant de s'approcher du détecteur de faute minimal pour le k-accord dans $\mathcal{AMP}$, il nous a semblé approprié de chercher à discerner les points communs aux détecteurs suffisants et ce qu'ils ajoutent à Σ_k que nous savions nécessaire.

Notre étude s'est alors portée, dans un premier temps, principalement sur $\mathcal{L}_k$ dont la spécification est très différente de celles des autres détecteurs connus. Ces investigations nous ont amené aux résultats présentés dans cette section qui lient $\mathcal{L}_k$ à certains autres détecteurs de fautes et qui visent à mieux cerner sa nature. Ces travaux ont été présentés dans un article [19] que nous avons soumis à la conférence SSS 2011.

3.1 Du réalisme de $\mathcal{L}_k$

Une des caractéristiques surprenantes de $\mathcal{L}_k$ est son aspect perpétuel très fort. En effet, si l'on fait l'analogie entre les processus qui se croient seuls et ceux qui sont considérés comme leaders dans les détecteurs de fautes basés sur une variante de Ω, il apparait que $\mathcal{L}_k$ n'autorise qu'au plus k processus (voir la propriété de sûreté) à être leaders (à se croire seuls) au cours d'une exécution.

Ce constat marque une différence fondamentale avec les détecteurs basés sur des variantes de Ω pour lesquels le nombre de leaders n'est pas limité pendant un temps arbitrairement long (la période d'anarchie signalée dans le chapitre précédent lors des définitions de Ω_k et de $\overline{\Omega}_k$).

Cette particularité a un impact sur une propriété des détecteurs de fautes que l'on nomme *réalisme* [12].

Réalisme d'un détecteur de faute

On dit d'un détecteur de faute qu'il est *réaliste* si on peut le simuler dans un système synchrone.

Définition de $\mathcal{SMP}$

On note par $\mathcal{SMP}$ un système distribué synchrone communicant par passage de messages. Dans un tel système, on considère que les processus sont équipés d'une horloge globale r

appelée *numéro de ronde*. Les algorithmes sont structurés autour de cette horloge globale : les processus progressent de façon synchrone de ronde en ronde ; au cours d'une ronde, ils envoient des messages à tous les autres processus, puis reçoivent les messages des autres processus et enfin exécutent des calculs locaux. Ce système fait une hypothèse importante sur les temps de livraison des messages : tout message envoyé lors d'une ronde est reçu au cours de la même ronde par les destinataires.

```
(01) init alone_i ← false ;
(02) when r = 1, 2, ... do
(03) begin synchronous round
(04)    broadcast ALIVE(i) ;
(05)    rec_ids_i ← { j such that ALIVE(j) received during the current round } ;
(06)    if (|rec_ids_i| ≤ n − k) then alone_i ← true end if
(07) end synchronous round.
```

Algorithme 1: $\mathcal{L}_k$ dans $\mathcal{SMP}$ avec $k \geq n/2$ (code pour p_i)

Lemme 1 *L'algorithme 1 implémente $\mathcal{L}_k$ dans $\mathcal{SMP}$ lorsque $k \geq \frac{n}{2}$.*

Preuve Montrons tout d'abord la propriété de sûreté. Soit r la première ronde où un processus passe $alone$ à $true$. Cela n'arrive que lorsqu'au moins k processus ont crashé avant la fin de leur phase d'envoi de la ronde r. En conséquence, après la fin de la phase d'envoi de la ronde r, il ne reste qu'au plus $n-k$ processus vivants. Comme $k \geq \frac{n}{2}$, on a $n-k \leq \frac{n}{2} \leq k$, et donc au plus k processus passeront leur variable $alone$ à $true$ (puisque, par définition de r, cela ne peut se produire avant la phase de calcul de cette ronde), ce qui prouve la propriété.

En ce qui concerne la propriété de vivacité, si k processus ou plus sont fautifs au cours d'une exécution, alors il existe une ronde r à laquelle les processus corrects recevront au plus $n-k$ messages. Ils passeront alors leur variable $alone$ à $true$ et la garderont à cette valeur pour toujours, ce qui prouve la propriété.

$\square_{Lemme\ 1}$

Lemme 2 *Il est impossible de simuler $\mathcal{L}_k$ dans $\mathcal{SMP}$ lorsque $k < \frac{n}{2}$.*

Preuve Supposons qu'il existe un algorithme qui construise $\mathcal{L}_k$ dans $\mathcal{SMP}$ avec $k < \frac{n}{2}$. Soit alors une exécution e_{k+1} où les processus $p_i, i \leq k$ crashent. Selon la propriété de vivacité de $\mathcal{L}_k$, un processus (p_{k+1} par exemple) doit alors passer $alone$ à $true$.

Considérons alors une exécution e_{k+2} similaire à e_{k+1} jusqu'à ce que p_{k+1} ait passé $alone$ à $true$ et dans laquelle p_{k+1} crashe ensuite. Toujours selon la propriété de vivacité, un autre processus (p_{k+2} par exemple) doit alors passer $alone$ à $true$. On répète alors la construction jusqu'à avoir construit une exécution r_n où $n-k$ processus ont passé $alone$ à $true$. Mais $k < \frac{n}{2}$ implique que $n-k > \frac{n}{2} > k$, la propriété de sûreté n'est donc pas vérifiée, ce qui contredit l'existence de notre algorithme.

$\square_{Lemme\ 2}$

Le théorème suivant découle des Lemmes 1 et 2.

Théorème 1 *$\mathcal{L}_k$ est réaliste si et seulement si $k \geq \frac{n}{2}$.*

3.2 De Σ_k à $\mathcal{L}_k$

Toujours pour tenter de mieux comprendre $\mathcal{L}_k$, nous avons cherché un détecteur de fautes équivalent dont la définition se rapprocherait de celle des autres détecteurs connus. Comme nous savions que Σ_k est plus faible que tout détecteur permettant le k-accord, donc plus faible que $\mathcal{L}_k$, nous avons déterminé une propriété à ajouter à celles de Σ_k, de manière à ce que le détecteur de fautes obtenu soit équivalent à $\mathcal{L}_k$.

3.2.1 Définition de Σ_k^+

Σ_k^+ est défini comme Σ_k et vérifie ses propriétés de vivacité et d'intersection. Il assure également la propriété suivante :

Solitude : $|\mathcal{F}| \geq k \implies \exists i \in \mathcal{C}, \exists \tau, \forall \tau' \geq \tau : qr_i^{\tau'} = \{i\}$

Ceci assure que dans le cas où plus de k processus sont fautifs, un des processus corrects finit par ne plus voir que lui dans son quorum.

3.2.2 Construction de $\mathcal{L}_k$ dans $\mathcal{AMP}[\Sigma_k^+]$

init $alone_i \leftarrow false$;
when $qr_i = \{i\} : alone_i \leftarrow true.$

Algorithme 2: $\mathcal{L}_k$ dans $\mathcal{AMP}[\Sigma_k^+]$

Lemme 3 *L'algorithme 2 implémente $\mathcal{L}_k$ dans $\mathcal{AMP}[\Sigma_k^+]$.*

Preuve La propriété d'intersection de Σ_k^+ garantit qu'au plus k processus p_i vérifieront $qr_i = \{i\}$ à un instant de l'exécution. Par conséquent, au plus k processus passeront $alone_i$ à $true$, la propriété de sûreté de $\mathcal{L}_k$ est donc vérifiée.

Selon la propriété de solitude de Σ_k^+, lorsqu'il y a plus de k processus fautifs au cours d'une exécution, un des processus correct se retrouve inéluctablement et pour toujours seul dans son quorum. Il passera donc sa variable $alone$ à $true$ et la gardera à cette valeur pour toujours. La propriété de vivacité de $\mathcal{L}_k$ est donc garantie.

$\square_{Lemme\ 3}$

3.2.3 Construction de Σ_k^+ dans $\mathcal{AMP}[\mathcal{L}_k]$

Lemme 4 *L'algorithme 3 implémente Σ_k^+ dans $\mathcal{AMP}[\mathcal{L}_k]$.*

Preuve

Vivacité et solitude Soit A l'ensemble des identités de processus qui lisent leur variable $alone$ à $true$ à un moment de l'exécution. À cause des lignes 02 et 03, tout processus $p_i, i \in A \cap \mathcal{C}$ passera sa variable qr_i à $\{i\}$ et la gardera à cette valeur pour toujours (conditions aux lignes 05 et 11). Il diffusera alors infiniment souvent des messages ALONE(i) (ligne 03). Nous considérons deux cas.

```
(01) init : qr_i ← P ; start T1, T2.

(02) when alone_i becomes equal to true :
(03)     qr_i ← {i} ; repeat forever broadcast ALONE(i) end repeat.

(04) when ALONE(j) with j ≠ i is received :
(05)     if (qr_i ≠ {i}) then qr_i ← {i, j} end if.

(06) task T1 : repeat forever broadcast ALIVE(i) end repeat.

(07) task T2 :
(08)     repeat forever
(09)         wait until (new ALIVE(j) messages with j ≠ i received from n − k processes) ;
(10)         let proc_i = the set of n − k processes from which messages have been received ;
(11)         if (qr_i ≠ {i}) then qr_i ← {i} ∪ proc_i end if
(12)     end repeat.
```

Algorithme 3: Σ_k^+ dans $\mathcal{AMP}[\mathcal{L}_k]$

- Cas 1 : il y a au moins k processus fautifs.
  Conformément à la propriété de vivacité de $\mathcal{L}_k$, $A \cap \mathcal{C} \neq \emptyset$, ainsi la propriété de solitude de Σ_k^+ est vérifiée.
  Après un certain temps (lorsque les processus de $A \cap \mathcal{F}$ ont crashé), seuls les processus de $A \cap \mathcal{C}$ envoient des messages ALONE. En conséquence, après un certain temps, les processus de $\mathcal{C} \setminus A$ ne recevront de messages ALONE qu'en provenance de processus de $A \cap \mathcal{C}$. Ils n'insèreront alors plus d'identités de processus fautifs dans leur quorum à la ligne 05.
  De plus, comme il y a plus de k processus fautifs, à partir d'un certain temps tous les processus corrects seront bloqués à la ligne 09 et n'exécuteront donc plus la ligne 11.
  Comme tout processus $p_i, i \in \mathcal{C} \setminus A$ affecte infiniment souvent à son quorum un ensemble $\{i, j\}$ avec $j \in A \cap \mathcal{C}$ (ligne 05) et n'y insère plus de processus fautifs à partir d'un certain temps, la propriété de vivacité de Σ_k^+ est vérifiée.
- Cas 2 : il y a au plus $k - 1$ processus fautifs.
  Dans ce cas, la propriété de solitude de Σ_k^+ découle directement de $|F| < k$.
  Comme il y a strictement moins de k processus fautifs, les processus corrects exécutent une infinité de fois la boucle des lignes 08 à 12. Il existe un instant après lequel seuls les processus corrects envoient des messages ALIVE. Donc, les processus de $\mathcal{C} \setminus A$ affectent infiniment souvent à leur quorum un ensemble contenant $n - k + 1$ processus corrects et n'insèrent finalement plus de processus fautifs dans leur quorum à la ligne 11.
  Par ailleurs, à partir d'un certain temps, seuls les processus de $A \cap \mathcal{C}$ (qui peut être vide) envoient des messages ALONE. Il existe donc un moment après lequel les processus corrects n'insèrent plus de processus fautifs dans leur quorum à la ligne 05.
  Après un certain temps, les processus de $\mathcal{C} \setminus A$ ont donc uniquement des processus corrects dans leur quorum, la propriété de vivacité de Σ_k^+ est donc garantie.

Intersection Supposons que l'on ait un ensemble $(q_i)_{1 \leq i \leq k+1}$ de $k + 1$ valeurs renvoyées par qr à des instants quelconques sur des processus quelconques tels que $\forall i \neq j : q_i \cap q_j = \emptyset$ (on note Q l'ensemble des identités des processus sur lesquels ces quorums ont été lus).

Comme tout processus est à tout moment dans son propre quorum, on a nécessairement $|Q| = k + 1$. On peut supposer sans perte de généralité que $Q = [\![1; k + 1]\!]$ et que q_i a été lu par p_i pour tout $i \in [\![1; k + 1]\!]$. Ces valeurs de quorums ont été affectées aux lignes 03, 05 ou 11 car la valeur initiale de qr ($\mathcal{P}$) intersecterait tout autre quorum.

Notons $Q_1 \subseteq Q$ l'ensemble des identités de processus qui ont lu une valeur calculée aux lignes 03 ou 05. Le quorum q_i lu par un processus $p_i, i \in Q_1$ contient l'identité d'un processus p_{x_i} qui a passé à un moment $alone_{x_i}$ à $true$. Comme $\forall i, j \in Q : i \neq j \implies q_i \cap q_j = \emptyset$, on doit avoir $\forall i, j \in Q_1 : i \neq j \implies x_i \neq x_j$. Selon la propriété de sûreté de $\mathcal{L}_k$, les processus qui passent $alone$ à $true$ sont au plus k. On en déduit que $1 \leq |Q_1| \leq k$ et donc que $Q \setminus Q_1 \neq \emptyset$.

Soit $i \in Q \setminus Q_1$. p_i a affecté son quorum avec q_i à la ligne 11, donc $|q_i| = n - k + 1$. Comme tout site est dans son propre quorum, $q_i \cap (Q \setminus \{i\}) = \emptyset$, et donc $|q_i| \leq |\mathcal{P}| - |Q \setminus \{i\}| = n - k$, ce qui est une contradiction. La propriété d'intersection de Σ_k^+ est donc vérifiée.

$\square_{Lemme\ 4}$

Le théorème suivant est la conséquence des Lemmes 3 et 4.

Théorème 2 $\Sigma_k^+ \simeq \mathcal{L}_k$

On dispose alors d'une propriété qui, ajoutée à Σ_k que l'on sait nécessaire, nous permet d'obtenir $\mathcal{L}_k$ qui est suffisant pour résoudre le k-accord. Cela permet de savoir quelles propriétés on peut ajouter à Σ_k sans qu'il devienne plus fort que $\mathcal{L}_k$, mais cela donne également des indications sur la propriété minimale à ajouter à Σ_k pour résoudre le k-accord. On sait désormais que cette propriété est contenue dans la notre.

3.3 $\Diamond\mathcal{L}_k \simeq \Omega_k$

Une autre approche qui nous a permis de mieux comprendre le fonctionnement du détecteur $\mathcal{L}_k$ est présentée ici. On peut faire l'analogie entre les processus qui vérifient $alone = true$ lorsqu'on les munit de $\mathcal{L}_k$ et ceux qui apparaissent dans les variables $leaders$ de Ω_k. La différence principale qui apparait est que, au cours d'une exécution donnée, au plus k processus peuvent avoir $alone = true$, alors qu'un nombre quelconque de processus peuvent être $leaders$, même s'ils finissent par n'être qu'au plus k.

Pour vérifier cette intuition, nous avons prouvé qu'en remplaçant la propriété de sûreté (perpétuelle) de $\mathcal{L}_k$ par son pendant inéluctable, nous obtenions un détecteur de fautes équivalent à Ω_k.

3.3.1 Définition de $\Diamond\mathcal{L}_k$

$\Diamond\mathcal{L}_k$ est défini comme $\mathcal{L}_k$ mais la propriété de sûreté est remplacée par la suivante :

Sûreté inéluctable : $\exists \tau : |\{i \in \mathcal{C} | \exists \tau' \geq \tau : alone_i^{\tau'}\}| \leq k$

Cette nouvelle version de la sûreté n'assure la sûreté de $\mathcal{L}_k$ qu'après un temps fini. Avant ce moment, plus de k processus peuvent passer leur variable $alone$ à $true$.

3.3.2 Construction de $\Diamond\mathcal{L}_k$ dans $\mathcal{AMP}[\Omega_k]$

Lemme 5 *L'algorithme 4 construit $\Diamond\mathcal{L}_k$ dans $\mathcal{AMP}[\Omega_k]$.*

init $alone_i \leftarrow false$;
repeat forever $alone_i \leftarrow (i \in leaders_i)$ **end repeat**.

Algorithme 4: $\diamond\mathcal{L}_k$ dans $\mathcal{AMP}[\Omega_k]$ (code pour p_i)

Preuve Soit τ un instant après lequel tous les processus fautifs ont crashés et tel qu'il existe un ensemble LD vérifiant $LD \cap \mathcal{C} \neq \emptyset \wedge \forall i \in \mathcal{C}, \forall \tau' \geq \tau : leaders_i^{\tau'} = LD$ (un tel instant existe, d'après la propriété d'autorité forte inéluctable de Ω_k). Les processus $p_i, i \in LD \cap \mathcal{C}$ exécutent infiniment souvent $alone_i \leftarrow true$, et n'exécutent plus jamais $alone_i \leftarrow false$ après τ. Par conséquent, la propriété de vivacité de $\diamond\mathcal{L}_k$ est vérifiée quel que soit le nombre de processus fautifs.

Les processus $p_i \in \mathcal{C} \setminus LD$ exécutent infiniment souvent $alone_i \leftarrow false$ et plus jamais $alone_i \leftarrow true$ après τ. Si on note $\tau' > \tau$ un instant tel que tous ces processus ont mis leur variable $alone$ à jour après τ et tous les processus fautifs ont crashé, on a alors au plus k processus corrects qui ont leur variable $alone$ à $true$ après τ'. La propriété de sûreté inéluctable de $\diamond\mathcal{L}_k$ est donc garantie.

$\square_{Lemme\ 5}$

3.3.3 Construction de Ω_k dans $\mathcal{AMP}[\diamond\mathcal{L}_k]$

(01) **init** $leaders_i \leftarrow \{1, 2, \ldots, k\}$; $r_i \leftarrow 0$; $next\_set_i \leftarrow \emptyset$;
(02) **repeat forever if** $(alone_i)$ **then** broadcast ALONE(i) **end if end repeat**.

(03) **when** ALONE(j) **is received** : **if** $(j \notin leaders_i)$ **then** broadcast NEXT$(r_i, leaders_i)$ **end if**.

(04) **when** NEXT$(r, leaders)$ **is received for the first time** :
(05) broadcast NEXT$(r, leaders)$;
(06) $next\_set_i \leftarrow next\_set_i \cup \{(r, leaders)\}$;
(07) **while** $\big((r_i, leaders_i) \in next\_set_i\big)$ **do**
(08) $(r_i, leaders_i) \leftarrow$ next_lead$(r_i, leaders_i)$;
(09) **end while**.

Algorithme 5: Ω_k dans $\mathcal{AMP}[\diamond\mathcal{L}_k]$ (code pour p_i)

Principe de l'algorithme

Dans cet algorithme, l'objectif est d'assurer qu'après un temps fini les processus corrects sont en accord sur le même ensemble de k leaders contenant au moins un d'entre-eux. Grâce à la propriété de sûreté inéluctable de $\mathcal{L}_k$, on sait que l'ensemble A des processus qui ont infiniment souvent leur variable $alone$ à $true$ est de cardinal au plus k. En assurant que les variables $leaders$ des processus corrects convergent toutes vers le même ensemble de k identités contenant A, on remplit notre objectif (s'il y a plus de k processus fautifs, $A \cap \mathcal{C} \neq \emptyset$, et sinon tout ensemble de k processus contient au moins un correct).

Une fonction pour parcourir les ensembles de k identités de processus

Considérons l'ensemble $\mathcal{P}_k$ des parties de $\mathcal{P}$ contenant k éléments. On peut définir un ordre total sur $\mathcal{P}_k$, par exemple à partir de l'ordre lexicographique en associant à chaque partie la liste triée de ses éléments. Dans ce cas, le plus petit élément de $\mathcal{P}_k$ est $p_{min} = \{1, 2, \ldots, k\}$, le second plus petit est $\{1, 2, \ldots, k-1, k+1\}$ et son plus grand élément est $p_{max} = \{n-k+1, \ldots, n\}$.

On définit la fonction next_lead comme suit :

$$\begin{array}{rcccll} \mathsf{next\_lead} & : & \mathbb{N} \times \mathcal{P}_k & \longrightarrow & \mathbb{N} \times \mathcal{P}_k & \\ & & (r,p) & \longmapsto & (r, succ(p)) & \textbf{if} \quad p \neq p_{max} \\ & & & & (r+1, p_{min}) & \textbf{if} \quad p = p_{max} \end{array}$$

où $succ(p)$ désigne le successeur de p selon l'ordre défini plus haut.

Ainsi, lorsque l'on exécute en boucle l'instruction $(r,p) \leftarrow \mathsf{next\_lead}(r,p)$, la variable p parcourt, dans l'ordre et de façon circulaire l'ensemble $\mathcal{P}_k$ et la variable r compte le nombre de tours effectués. Finalement, la paire (r,p) parcourt $\mathbb{N} \times \mathcal{P}_k$ selon l'ordre lexicographique défini par :

$$(r,p) < (r',p') \Leftrightarrow (r < r') \vee ((r = r') \wedge (p < p'))$$

Comportement d'un processus p_i

La tâche de la ligne 02 assure qu'un processus informe régulièrement les autres par l'envoi de messages ALONE lorsque sa variable $alone_i$ vaut $true$. Lorsqu'il reçoit un tel message (ligne 03), un processus s'assure que l'identité transportée par le message apparait bien dans sa variable $leaders_i$. Si ce n'est pas le cas, il informe les processus (y compris lui-même) que l'état convergé n'est pas atteint en diffusant un message NEXT qui porte la valeur courante de son ensemble $leaders_i$ ainsi que le nombre de parcours de $\mathcal{P}_k$ que sa variable $leaders_i$ a effectués jusque là.

Le traitement des messages NEXT se fait en plusieurs étapes. Ils sont d'abord réémis (ligne 05) aux autres processus (mécanisme de diffusion fiable) afin que tous les processus corrects reçoivent les mêmes messages (même lorsqu'un processus cesse de fonctionner au milieu d'une opération de diffusion). La paire de valeurs qu'ils transportent est sauvegardée dans $next\_set_i$ avec toutes les autres reçues jusque là (ligne 06). Ensuite, si le message portait les valeurs courantes de $leaders_i$ et de r_i, ces dernières sont mises à jour grâce à la fonction next_lead. Enfin, tant que les nouvelles valeurs de $leaders_i$ et de r_i correspondent à une des paires sauvegardées dans $next\_set_i$ on continue à les faire progresser (lignes 07 à 09).

Ainsi, grâce au mécanisme de diffusion fiable, on assure que lorsque plus aucun message NEXT n'est émis, tous les processus corrects finissent par mettre le même ensemble dans leur variable $leaders$ et ne la modifient plus.

Lemme 6 *Soit p_i un processus correct et τ un instant. Si $r_i^\tau = r$ et $leaders_i^\tau = ld$, alors le processus p_i a reçu et diffusé tous les messages* NEXT*(r', ld') tels que $(r', ld') < (r, ld)$ avant τ.*

Preuve Les seules modifications des variables $leaders_i$ et r_i ont lieu à la ligne 08. La définition de la fonction next_lead et la valeur initiale de la paire $(r_i, leaders_i)$ impliquent que lorsque

l'on a $(r_i^\tau, leaders_i^\tau) = (r, ld)$, alors la paire $(r_i, leaders_i)$ a pris toutes les valeurs (r', ld') telles que $(0, p_{min}) \leq (r', ld') < (r, ld)$ à des instants avant τ.

On remarque que la paire $(r_i, leaders_i)$ n'est modifiée que lorsque sa valeur courante apparait dans $next\_set_i$ (ligne 07). Or toutes les paires qui apparaissent dans l'ensemble $next\_set_i$ ont été au préalable reçues et rediffusées (lignes 04 à 06). Ainsi toutes les valeurs (r', ld') qui ont été prises avant τ par la paire $(r_i, leaders_i)$ (exceptée la valeur courante (r, ld)) correspondent à celles transportées par un des messages NEXT qui a été reçu et rediffusé par p_i avant τ.

$\square_{Lemme\ 6}$

Lemme 7 *L'algorithme 5 implémente Ω_k dans $\mathcal{AMP}[\diamond\mathcal{L}_k]$.*

Preuve Tout d'abord, grâce à l'initialisation de $leaders_i$ et à la définition de la fonction next_lead, $leaders_i$ contient toujours un ensemble de k identités de processus et la propriété de validité de Ω_k est vérifiée.

Soit Π l'ensemble des identités des processus qui diffusent une infinité de messages ALONE au cours de l'exécution. Selon la propriété de sûreté de $\diamond\mathcal{L}_k$, les processus qui voient une infinité de fois la variable $alone_i$ à $true$ sont au plus k. Le fait qu'un processus n'envoie un message ALONE que lorsqu'il vient de lire $alone_i$ à $true$ (ligne 02) entraine que $|\Pi| \leq k$.

Il y a un instant après lequel (a) tous les messages ALONE reçus proviennent de processus de Π_1 et (b) aucun processus ne diffuse plus de message NEXT(r,ld) avec $ld \supseteq \Pi$ (à cause de la condition de la ligne 03). Soit S l'ensemble des paires (r, ld) vérifiant (a) $ld \supseteq \Pi$ et (b) un message NEXT(r,ld) a provoqué la mise à jour de la paire (r_i,$leaders_i$) d'un processus correct p_i. Conformément à la remarque précédente, S est un ensemble fini. Si $S \neq \emptyset$, on peut donc définir (r_0, ld_0) comme la plus grande paire de S. Sinon, pour préserver la généralité du raisonnement suivant, on suppose alors que (r_0, ld_0) correspond à une paire strictement plus petite que toutes celles de $\mathbb{N} \times \mathcal{P}_k$ et telle que next_lead$(r_0, ld_0) = (0, p_{min})$.

Soit (r_1, ld_1) la plus petite paire telle que $(r_0, ld_0) < (r_1, ld_1)$ et $ld_1 \supseteq \Pi_1$. De la définition de (r_0, ld_0) et du Lemme 6, on déduit qu'un processus correct a reçu et diffusé des messages NEXT(r, ld) pour toute paire $(r, ld) \leq (r_0, ld_0)$. Tous les processus corrects finiront donc par exécuter $(r_i, ld_i) \leftarrow$ next_lead(r_0, ld_0).

Lorsque sur un processus correct p_i à τ on a $leaders_i^\tau \not\supseteq \Pi_1$, conformément à la définition de Π_1, p_i finit par recevoir un message ALONE(j) avec $j \in \Pi_1 \setminus leaders_i^\tau$. À la réception d'un tel message, si la valeur de $leaders_i$ n'a pas changé, p_i diffuse un message NEXT($-$,$leader_i$) (ligne 03), finit par le recevoir et met donc à jour sa paire $(r_i, leaders_i)$. Par conséquent, pour toute paire (r, ld) telle que $(r_0, ld_0) < (r, ld) < (r_1, ld_1)$, comme $ld \not\supseteq \Pi_1$, un processus correct diffuse un message NEXT(r, ld). Une fois que ces messages ont été reçus et traités par tous les processus corrects, ils ont tous les valeurs (r_1, ld_1) dans leur paire de variables $(r_i, leaders_i)$ et ne la mettent plus à jour (sinon on contredit les définitions de (r_0, ld_0) et de (r_1, ld_1)).

Dans le cas où il y a au moins k défaillances au cours de l'exécution, selon la propriété de vivacité de $\mathcal{L}_k$, $\Pi \cap \mathcal{C} \neq \emptyset$. Or $ld_1 \supseteq \Pi$ donc ld_1 contient bien au moins un processus correct. Dans le cas où il y a au plus $k-1$ défaillances, tout ensemble de k éléments contient au moins un processus correct, donc c'est vrai pour ld_1. Ainsi, tous les processus corrects finissent par s'accorder sur un même ensemble de leaders contenant un d'entre eux. La propriété d'autorité forte inéluctable de Ω_k est donc vérifiée.

$\square_{Lemme\ 7}$

Après cette mise en relation de $\mathcal{L}_k$ avec les détecteurs des classes Σ_k et Ω_k, on comprend mieux ce détecteur de fautes. $\mathcal{L}_k$ est une version de Σ_k dans laquelle, lorsqu'il y a au moins k défaillances un processus doit se croire seul (on comprend alors pourquoi il est optimal lorsque $k = n - 1$ mais pas dans les autres cas). Par ailleurs, $\mathcal{L}_k$ est une version perpétuelle de Ω_k. Comme dans Ω_k la découverte des leaders ne se fait toujours qu'après un temps arbitrairement long, cependant, au plus k processus sont considérés comme leaders au cours d'une exécution.

Chapitre 4

Renforcement d'un détecteur : de Π_k à $\Box\Pi_k$

Après notre étude de $\mathcal{L}_k$ et de Ω_k, il apparait que $\mathcal{L}_k \succeq \Omega_k$ et que $\mathcal{L}_k$ est une version perpétuelle de Ω_k. Par ailleurs, $\mathcal{L}_k$ étant suffisant pour résoudre le k-accord dans $\mathcal{AMP}$, on sait que $\mathcal{L}_k \succeq \Sigma_k$. On a donc $\mathcal{L}_k \succeq \langle \Sigma_k, \Omega_k \rangle$. Or cette paire de détecteurs de fautes a déjà été étudiée, elle est équivalente à Π_k [3]. $\mathcal{L}_k$ peut donc également être vu comme une variante perpétuelle de Π_k.

Π_k correspond aux détecteurs de fautes optimaux pour les cas $k = 1$ et $k = n-1$, cependant, il a été prouvé dans [5] qu'il était impossible de résoudre le k-accord dans $\mathcal{AMP}[\Pi_k]$ dès que $k > 1$ et $2k^2 \leq n$, il n'est donc pas suffisant pour résoudre ce problème dans le cas général. Il paraissait alors intéressant d'essayer de trouver une autre modification à apporter à Π_k pour obtenir un détecteur suffisant pour le k-accord.

Ce chapitre introduit $\Box\Pi_k$, un détecteur de fautes issu de cette recherche, présente un algorithme qui résoud le k-accord dans $\mathcal{AMP}[\Box\Pi_k]$ et montre que ce détecteur est strictement plus faible que deux autres détecteurs adaptés au k-accord mais incomparables. De plus, $\Box\Pi_k$ est réaliste.

4.1 Définition de $\Box\Pi_k$

$\Box\Pi_k$ est défini de la même façon que Π_k, seule la propriété d'autorité inéluctable est modifiée et rendue perpétuelle (la propriété d'intersection devient alors redondante et peut être omise). $\Box\Pi_k$ fournit à chaque processus une variable *quorum* notée qr contenant un ensemble d'identités de processus. $\Box\Pi_k$ vérifie la propriété de vivacité de Σ_k et une propriété d'autorité perpétuelle additionnelle :

Vivacité : $\exists \tau : \forall i \in \mathcal{C}, \forall \tau' \geq \tau : qr_i^{\tau'} \subseteq \mathcal{C}$

Autorité perpétuelle : $\exists LD \subseteq \mathcal{P} : (|LD| = k) \wedge (\forall i \in \mathcal{P}, \forall \tau : qr_i^{\tau} \cap LD \neq \emptyset)$

Les quorums des processus corrects finissent par ne plus contenir que des processus corrects. La propriété d'autorité perpétuelle assure qu'il y a un ensemble de k leaders tel que l'on trouve au moins l'un d'entre eux dans chaque quorum. On remarquera que la propriété d'intersection de Σ_k découle de la propriété d'autorité perpétuelle (ce n'était pas le cas avec la propriété d'autorité inéluctable de Π_k).

4.2 $\square\Pi_k$ est suffisant pour le k-accord

```
when operation set_agreement_k(v_i) is invoked by p_i
(01) for r_i from 1 to (n k) do
(02)    broadcast EST(i, r_i, v_j)
(03)    wait until for all j ∈ qr_i, an EST(j, r_i, v_j) message has been received
(04)    if ∃j ∈ round_leaders(r_i) : EST(j, r_i, v_j) received
(05)       then v_i ← v_j
(06)    end if
(07) end for
(08) return v_i
```

Algorithme 6: k-accord dans $\mathcal{AMP}[\square\Pi_k]$ (code pour p_i)

Théorème 3 *L'algorithme 6 résoud le k-accord dans $\mathcal{AMP}[\square\Pi_k]$.*

Preuve Précisons tout d'abord que la fonction $round\_leaders$ peut être n'importe quelle bijection de l'ensemble des entiers de 1 à $\binom{n}{k}$ sur l'ensemble des parties à k éléments de $\mathcal{P}$.

Notons V_r l'ensemble des valeurs des variables v_i lorsque p_i termine la r^{e} itération de la boucle des lignes 01 à 07 pour tout $i \in \mathcal{P}$ tel que p_i termine cette r^{e} itération. Soit V_0 l'ensemble des valeurs proposées par les processus. Lorsque la variable v_i d'un processus est modifiée au cours d'une itération, elle prend la valeur qu'avait une variable v_j d'un processus p_j lorsqu'il a commencé cette même itération (lignes 02 à 05). Par conséquent, on a : $\forall r \in [\![1; \binom{n}{k}]\!] : V_r \subseteq V_{r-1}$. Il vient alors que $V_{\binom{n}{k}} \subseteq V_0$ et la propriété de validité du k-accord en découle.

Pour prouver la terminaison, il suffit de montrer qu'un processus correct ne peut rester bloqué sur l'attente de la ligne 03. Supposons que des processus corrects restent bloqués sur cette ligne. Soit alors r_{min} le plus petit indice tel qu'un processus correct p_i bloque à la ligne 03 au cours de la ${r_{min}}^{\text{e}}$ itération de sa boucle. Comme aucun processus correct n'est bloqué dans une itération $r < r_{min}$, tout processus correct p_j exécute la ligne 02 à la ${r_{min}}^{\text{e}}$ itération. Il diffuse donc un message $\text{EST}(j, r_{min}, v_j)$ et p_i finit donc par le recevoir. p_i reçoit ainsi inéluctablement un message $\text{EST}(j, r_{min}, v_j)$ de chaque processus correct p_j. Or, selon la propriété de vivacité des quorums, il existe un moment à partir duquel qr_i ne contient que des identités de processus corrects. Donc à partir d'un certain temps, qr_i ne contient plus que des identités de processus dont p_i a reçu un message $\text{EST}(j, r_{min}, v_j)$, ce qui contredit le fait que p_i soit bloqué à la ligne 03 au cours de la ${r_{min}}^{\text{e}}$ itération de sa boucle. L'algorithme assure donc la propriété de terminaison.

Soit LD un ensemble tel que décrit dans la propriété d'autorité perpétuelle de $\square\Pi_k$ et soit r_{LD} l'entier entre 1 et $\binom{n}{k}$ tel que $round\_leaders(r_{LD}) = LD$. Comme tous les processus qui décident ont auparavant exécuté toute la boucle des lignes 01 à 07, ils en ont, en particulier, exécuté la ${r_{LD}}^{\text{e}}$ itération. Lors de cette itération, pour tous les processus, la condition de la ligne 04 est vérifiée, car la propriété d'autorité perpétuelle implique que le quorum qui a permis la fin de l'attente à la ligne 03 contient au moins un des éléments de LD. Par conséquent, lorsqu'ils exécutent la ${r_{LD}}^{\text{e}}$ itération de leur boucle, tous les processus adoptent une des valeurs transportées par les messages diffusés par les processus de LD lors de leur ${r_{LD}}^{\text{e}}$ itération. On a donc :

$$V_{r_{LD}} \subseteq \{v_j \mid \exists j \in LD : p_j \text{ a envoyé } \text{EST}(j, r_{LD}, v_j) \text{ lors de sa } {r_{LD}}^{\text{e}} \text{ itération}\}$$

on a, grâce à la décroissance des ensembles V_r montrée en début de preuve, $V_{\binom{n}{k}} \subseteq V_{r_{LD}}$. Or $|LD| = k$, donc les processus de LD qui diffusent un message EST à la $r_{LD}{}^{\text{e}}$ itération sont au plus k. Il vient alors $|V_{r_{LD}}| \leq k$ et donc $|V_{\binom{n}{k}}| \leq k$, ce qui prouve que l'algorithme vérifie la propriété d'accord.

$\square_{Théorème\ 3}$

On dispose donc désormais d'une version perpétuelle du couple $\langle \Sigma_k, \Omega_k \rangle$ qui permet toujours la résolution du k-accord.

4.3 $\square\Pi_k \prec \mathcal{L}_k$

Après avoir prouvé que $\square\Pi_k$ est suffisant pour résoudre le k-accord dans $\mathcal{AMP}$, tâchons maintenant de le comparer à $\mathcal{L}_k$ qui est lui même une version perpétuelle de Ω_k.

4.3.1 Construction de $\square\Pi_k$ dans $\mathcal{AMP}[\mathcal{L}_k]$

L'algorithme utilisé pour construire $\square\Pi_k$ dans $\mathcal{AMP}[\mathcal{L}_k]$ est l'algorithme 3 qui permettait de construire Σ_k^+ dans $\mathcal{AMP}[\mathcal{L}_k]$.

Lemme 8 *L'algorithme 3 implémente $\square\Pi_k$ dans $\mathcal{AMP}[\mathcal{L}_k]$.*

Preuve La preuve de la propriété de vivacité est la même que dans la preuve du Lemme 4. Pour une exécution donnée, soit LD un ensemble de k identités de processus qui contient tous les processus qui passent leur variable $alone$ à $true$. Un quorum qui n'a jamais été modifié depuis son initialisation est égal à $\mathcal{P}$, il contient donc un élément de LD. Un quorum dont la valeur a été affectée aux lignes 03 ou 05 contient un processus qui a passé $alone$ à $true$, il contient aussi un élément de LD. Enfin, un quorum dont la valeur a été calculée à la ligne 11 contient $n - k + 1$ identités de processus, il contient par conséquent également un des éléments de LD. La propriété d'autorité perpétuelle est donc vérifiée par cet algorithme.

$\square_{Lemme\ 8}$

4.3.2 $\square\Pi_k \not\succeq \mathcal{L}_k$

Lemme 9 *Il est impossible d'implémenter $\mathcal{L}_k$ dans $\mathcal{AMP}[\square\Pi_k]$ si $k < n - 1$.*

Preuve Supposons que l'on dispose d'un algorithme A qui implémente $\mathcal{L}_k$ dans $\mathcal{AMP}[\square\Pi_k]$. Considérons pour tout $i \in [\![1; k-1]\!]$ une exécution e_i dans laquelle tous les processus hormis p_i crashent initialement. $\forall \tau : qr_i^\tau = \{i\}$ est une sortie valide de $\square\Pi_k$ dans cette exécution. Comme A implémente $\mathcal{L}_k$, selon la propriété de vivacité de $\mathcal{L}_k$, il existe un instant τ_i après lequel la variable $alone_i$ garde la valeur $true$ pour toujours.

Soit alors $\tau_{max} = \max\{\tau_i \mid i \in [\![1; k-1]\!]\}$ et soit e une exécution dans laquelle p_k est crashé initiallement et tous les messages sont retardés jusquà τ_{max}.

On suppose que dans l'exécution e, les sorties de $\square\Pi_k$ sont les suivantes à tout instant τ :

– $\forall i \in [\![1; k-1]\!] : qr_i^\tau = \{i\}$

– $qr_k^\tau = \mathcal{P}$

– $qr^{\tau}_{k+1} = \mathcal{P} \setminus [\![1; k]\!]$
– $\forall i \in [\![k+2; n]\!] : qr^{\tau}_i = \mathcal{P} \setminus [\![1; k+1]\!]$

On remarquera que la propriété d'autorité perpétuelle est vérifiée avec $LD = [\![1; k-1]\!] \cup \{k+2\}$.

Pour tout $i \in [\![1; k-1]\!]$, e et e_i sont indistinguables du point de vue de p_i. En conséquence, dans e à τ_{max}, tous les processus $p_i, i \in [\![1; k-1]\!]$ ont passé leur variable $alone_i$ à $true$.

On suppose alors que dans e les processus $p_i, i \in [\![1; k-1]\!]$ crashent après τ_{max}. Comme A implémente $\mathcal{L}_k$ et qu'il y a k processus défaillants dans e, un autre processus (disons p_{k+1}) doit alors passer sa variable $alone$ à $true$. Supposons maintenant qu'après avoir passé $alone_{k+1}$ à $true$, p_{k+1} crashe (possible si $k < n-1$). Un autre processus doit alors passer sa variable $alone$ à $true$ pour satisfaire la propriété de vivacité de $\mathcal{L}_k$. Cependant, cela serait en contradiction avec la propriété de sûreté de $\mathcal{L}_k$ car $k+1$ processus auraient alors passé leur variable $alone$ à $true$ au cours de e.

Il est donc impossible pour A de fournir une sortie correcte pour $\mathcal{L}_k$ lors de l'exécution e. Les propriétés de vivacité et d'autorité perpétuelle de $\Box\Pi_k$ étant bien vérifiées par la sortie fournie pour e, ceci montre qu'il est impossible de simuler $\mathcal{L}_k$ dans $\mathcal{AMP}[\Box\Pi_k]$.

$\Box_{Lemme\ 9}$

On remarquera que, comme (a) $\Box\Pi_{n-1}$ permet de résoudre le $n-1$-accord (Lemme 8) et (b) $\mathcal{L}_{n-1}$ est le détecteur minimal pour le $n-1$-accord dans $\mathcal{AMP}$, il en découle que $\Box\Pi_{n-1} \succeq \mathcal{L}_{n-1}$. Le théorème suivant est la conséquence de cette observation et des Lemmes 8 et 9.

Théorème 4 *Pour tout $k < n-1$, $\Box\Pi_k \prec \mathcal{L}_k$. Par ailleurs, $\Box\Pi_{n-1} \simeq \mathcal{L}_{n-1}$.*

On a ainsi montré que notre nouveau détecteur de faute permet de résoudre le k-accord avec moins d'information que n'en apporte $\mathcal{L}_k$.

4.4 $\Box\Pi_k \prec \mathcal{S}_{n-k+1}$

Dans [1], il est montré que (a) $\mathcal{L}_1 \succ \mathcal{S}_n$, (b) $\forall k > 1 : \mathcal{L}_k \not\succeq \mathcal{S}_{n-k+1}$, (c) $\forall k < n-1 : \mathcal{L}_k \not\preceq \mathcal{S}_{n-k+1}$ et (d) $\mathcal{L}_{n-1} \prec \mathcal{S}_2$. $\mathcal{L}_k$ et $\mathcal{S}_{n-k+1}$ sont donc incomparables dans le cas général, par ailleurs ils sont tous deux suffisants pour résoudre le k-accord dans $\mathcal{AMP}$ [1]. Nous montrons dans cette section que $\Box\Pi_k$ est aussi plus faible que $\mathcal{S}_{n-k+1}$.

4.4.1 Construction de $\Box\Pi_k$ dans $\mathcal{AMP}[\mathcal{S}_{n-k+1}]$

Lemme 10 *Un algorithme qui exécute répétivement, sur tous les processus p_i, l'instruction $qr_i \leftarrow (\mathcal{P} \setminus suspected_i) \cup \{i\}$ simule $\Box\Pi_k$ dans $\mathcal{AMP}[\mathcal{S}_{n-k+1}]$.*

Preuve Selon la propriété de complétude de $\mathcal{S}_{n-k+1}$, à partir d'un certain temps, tous les processus fautifs sont toujours suspectés par les processus corrects. En conséquence, il existe un instant après lequel les processus corrects n'ont plus de processus fautifs dans leur quorum. La propriété de vivacité de $\Box\Pi_k$ est donc vérifiée.

Soit S un ensemble tel que décrit dans la propriété de $n-k+1$-précision de $\mathcal{S}_{n-k+1}$. Il y a donc un processus $p_\ell, \ell \in S \cap C$ qui n'est jamais suspecté par aucun des $n-k+1$ processus de

S. Soit alors $LD = (\mathcal{P} \setminus S) \cup \{\ell\}$. Comme $|S| = n-k+1$ et $\ell \in S$, $|LD| = k$. p_ℓ n'est jamais suspecté par les processus de S, donc ℓ est dans tous les quorums des processus de S. Ces quorums contiennent donc tous un élément de LD. Par ailleurs, $\forall i \in \mathcal{P} \setminus S, \forall \tau : i \in qr_i^\tau \cap LD$, donc tous les quorums des processus de $\mathcal{P} \setminus S$ contiennent également un élément de LD. La propriété d'autorité perpétuelle est donc vérifiée.

$\square_{Lemme\ 10}$

4.4.2 $\square\Pi_k \not\succeq \mathcal{S}_{n-k+1}$

Lemme 11 *Si $k > 1$, il n'existe pas d'algorithme qui simule $\mathcal{S}_{n-k+1}$ dans $\mathcal{AMP}[\square\Pi_k]$.*

Preuve Supposons qu'il existe un algorithme A qui implémente $\mathcal{S}_{n-k+1}$ dans $\mathcal{AMP}[\square\Pi_k]$ lorsque $1 < k < n-1$. Pour toutes les exécutions considérées, on supposera que $\square\Pi_k$ fournit les valeurs suivantes à tout instant τ :

- $\forall i \in [\![1; k-1]\!] : qr_i^\tau = \{i\}$
- $qr_{k+1} = \{k-1, k+1\}$
- $qr_{k+2} = \{k, k+2\}$
- $\forall i > k+2 : qr_i^\tau = \mathcal{P}$

Soient $e_i, i \in [\![1; k]\!]$ des exécutions de A dans lesquelles tous les processus sauf p_i sont initialement crashés. Dans e_i, selon les propriétés de $\mathcal{S}_{n-k+1}$, il existe τ_i tel que $\forall \tau \geq \tau_i$: $suspected_i^{\tau_i} = \mathcal{P} \setminus \{i\}$.

Soient $\tau_{max_1} = \max\{\tau_i \mid i \in [\![1; k]\!]\}$ et e_{k+1} une exécution dans laquelle (a) tous les messages sont retardés jusqu'à τ_{max_1} et (b) tous les processus $p_i, i \notin \{k-1, k+1\}$ crashent juste après τ_{max_1}. Dans e_{k+1}, il existe un instant τ_{k+1} tel que $\forall \tau \geq \tau_{k+1}$: $suspected_{k+1}^\tau \supseteq \mathcal{P} \setminus \{k-1, k+1\}$. Construisons également e_{k+2} dans laquelle (a) tous les messages sont retardés jusqu'à τ_{max_1} et (b) tous les processus $p_i, i \notin \{k, k+2\}$ crashent juste après τ_{max_1}. Dans e_{k+2}, il existe un instant τ_{k+2} tel que $\forall \tau \geq \tau_{k+2} : suspected_{k+2}^\tau \supseteq \mathcal{P} \setminus \{k, k+2\}$.

Soit alors e une exécution dans laquelle (a) tous les messages sont retardés jusqu'à τ_{max_1}, (b) les messages envoyés après τ_{max_1}, sauf ceux échangés entre p_{k-1} et p_{k+1} ou entre p_k et p_{k+2}, sont retardés jusqu'à $\tau_{max_2} = \max\{\tau_{k+1}, \tau_{k+2}\}$. Pour tout $i \in [\![1; k]\!]$, e et e_i sont indistinguables du point de vue de p_i. Par conséquent, dans e on a également $suspected_i^{\tau_i} = \mathcal{P} \setminus \{i\}$. Pour $i \in \{k-1, k+1\}$, e est indistinguable de e_{k+1} du point de vue de p_i. Donc, dans e, on a aussi $suspected_{k+1}^{\tau_{k+1}} \supseteq \mathcal{P} \setminus \{k-1, k+1\}$. Pour $i \in \{k, k+2\}$, e est indistinguable de e_{k+2} du point de vue de p_i. Ainsi $suspected_{k+2}^{\tau_{k+2}} \supseteq \mathcal{P} \setminus \{k, k+2\}$ reste vrai dans e.

On constate que la sortie fournie pour $\square\Pi_k$ est correcte dans toutes les exécutions considérées (avec $LD = [\![1; k]\!]$). Cependant, la propriété de $n-k+1$-précision ne peut pas être vérifiée dans e. En effet, comme tout processus $p_i, i \in [\![1; k]\!]$ suspecte tout le monde sauf lui-même à τ_i, au plus un d'entre eux peut avoir son identité dans S (il doit y avoir un processus qui n'est jamais suspecté par aucun processus de S). On a donc $|S \cap [\![1; k]\!]| \leq 1$, or $|\mathcal{P} \setminus [\![1; k]\!]| = n-k$ et $|S| = n-k+1$, par conséquent $|S \cap [\![1; k]\!]| = 1$ et $\{k+1, k+2\} \subseteq S$. Mais $suspected_{k+1}^{\tau_{k+1}} \supseteq \mathcal{P} \setminus \{k-1, k+1\}$ et $suspected_{k+2}^{\tau_{k+2}} \supseteq \mathcal{P} \setminus \{k, k+2\}$, donc $\mathcal{P} \setminus (suspected_{k+1}^{\tau_{k+1}} \cup suspected_{k+2}^{\tau_{k+2}}) = \emptyset$, ce qui montre qu'il n'y a pas de processus qui ne soit jamais suspecté ni par p_{k+1} ni par p_{k+2}. Cela contredit l'existence de l'algorithme A.

Par ailleurs, il est montré dans [1] que $\mathcal{L}_{n-1} \prec \mathcal{S}_2$. Or nous avons montré dans le théorème 4 que $\square\Pi_{n-1} \simeq \mathcal{L}_{n-1}$. Il vient alors que $\square\Pi_{n-1} \prec \mathcal{S}_2$ ce qui prouve le lemme dans le cas $k = n-1$ et qui conclut sa preuve.

$\square_{Lemme\ 11}$

Lemme 12 *Un algorithme qui exécute répétivement sur tout processus* p_i $suspected_i \leftarrow \mathcal{P} \setminus (qr_i \cup \{i\})$ *simule* $\mathcal{S}_n$ *dans* $\mathcal{AMP}[\square\Pi_1]$.

Preuve La propriété de vivacité de $\square\Pi_1$ assure qu'à partir d'un certain temps on a toujours, pour tout processus $p_i, i \in \mathcal{C}$, $qr_i \subseteq \mathcal{C}$. Par conséquent, après cet instant, $qr_i \cup \{i\} \subseteq \mathcal{C}$, $\mathcal{P} \setminus (qr_i \cup \{i\}) \supseteq \mathcal{P} \setminus \mathcal{C} = \mathcal{F}$. Ainsi, après un certain temps, on a toujours $\mathcal{F} \subseteq suspected_i$, la propriété de complétude est donc vérifiée.

Soit LD un ensemble tel que défini dans la propriété d'autorité perpétuelle de $\square\Pi_1$. Comme tous les quorums fournis par $\square\Pi_1$ contiennent l'unique élément de LD, le processus (correct) qui porte cette identité n'est jamais suspecté par aucun processus. La propriété de précision de $\mathcal{S}_n$ est donc garantie.

$\square_{Lemme\ 12}$

Le théorème suivant découle des Lemmes 10 et 11.

Théorème 5 *Pour* $k > 1$, *on a* $\square\Pi_k \prec \mathcal{S}_{n-k+1}$. *Par ailleurs,* $\square\Pi_1 \simeq \mathcal{S}_n$.

Du fait que $\mathcal{S}_n$ est implémentable dans un système synchrone et du théorème 5, il vient le corollaire suivant :

Corollaire 1 *Pour* $k \in [\![1; n-1]\!]$, $\square\Pi_k$ *est réaliste.*

Finalement, notre nouveau détecteur de faute $\square\Pi_k$ est plus faible que $\mathcal{L}_k$ et que $\mathcal{S}_{n-k+1}$. Ces détecteurs de fautes étant incomparables, nous avons réussi à en extraire une partie commune qui permet toujours de résoudre le k-accord. Cependant, $\square\Pi_k$ a également une propriété perpétuelle qui fait qu'il ne peut pas être plus faible que des détecteurs ($\langle \Sigma_x, \overline{\Omega}_y \rangle$ ou Σ_z par exemple) qui n'exigent pas de propriété perpétuelle autre que la propriété d'intersection de Σ. Cela explique par exemple que $\square\Pi_1 \simeq \mathcal{S}_n \succ \langle \Sigma, \Omega \rangle$, et donc que $\square\Pi_k$ n'est pas optimal lorsque $k = 1$.

Chapitre 5

Affaiblissement d'une des familles existantes : de $\langle \Sigma_x, \overline{\Omega}_y \rangle$ à $\Pi\Sigma_{x,y}$

Après avoir étudié $\Box\Pi_k$, nous avions trouvé un détecteur de faute suffisant pour le k-accord mais que son côté perpétuel éliminait de la course pour le détecteur minimal. Nous nous sommes alors penché sur la famille de détecteurs $\langle \Sigma_x, \overline{\Omega}_y \rangle$qui est montrée suffisante pour résoudre le xy-accord [5] et dont la seule propriété perpétuelle est celle de Σ_x.

Nous présentons ici $\Pi\Sigma_{x,y}$, un détecteur de faute dont le comportement s'approche de celui de la paire $\langle \Sigma_x, \overline{\Omega}_y \rangle$. $\Pi\Sigma_{x,y}$ tente de réunir les deux composants de la paire $\langle \Sigma_x, \overline{\Omega}_y \rangle$ en faisant dépendre la propriété d'autorité de Ω_y des quorums fournis par Σ_x.

Nous montrons dans ce chapitre que nous obtenons ainsi un détecteur qui permet de résoudre le k-accord d'une manière similaire à celle employée dans [5] mais en se basant sur un détecteur en général plus faible.

Pour conclure ce chapitre et ce rapport, nous prouvons, grâce à ce nouveau détecteur, que l'on peut résoudre le k-accord sans Ω_k, ce qui permettra de mieux cibler les futures investigations pour trouver le détecteur minimal pour le k-accord. Le contenu de ce chapitre est la base d'un article en cours de rédaction.

5.1 Définition de $\Pi\Sigma_{x,y}$

La définition de $\Pi\Sigma_{x,y}$ sera faite en deux temps. Nous définissons tout d'abord $\Pi\Sigma_x$, qui correspond à $\Pi\Sigma_{x,1}$, puis nous construisons $\Pi\Sigma_{x,y}$ en utilisant la définition de $\Pi\Sigma_x$.

5.1.1 $\Pi\Sigma_x$

$\Pi\Sigma_x$ est un détecteur de faute dont la définition ressemble à celle de la paire $\langle \Sigma_x, \overline{\Omega}_1 \rangle$. La principale différence est que $\Pi\Sigma_x$ n'impose plus que tous les processus finissent par s'accorder sur un même leader correct, il assure uniquement qu'un processus correct deviendra le leader de tous les processus dont les quorums sont infiniment souvent en intersection avec le sien.

Intuitivement, les propriétés de Σ_x garantissent que les processus ne sont pas séparés en plus de x parties, et la propriété d'autorité impose qu'un processus correct finisse par devenir le leader de toute sa partie.

Formellement, $\Pi\Sigma_x$ fournit aux processus une variable qr contenant un ensemble d'identités de processus et une variable ld contenant une identité de processus. Les propriétés suivantes doivent être vérifiées :

Vivacité : $\exists \tau : \forall \tau' \geq \tau, \forall i \in \mathcal{C} : qr_i^{\tau'} \subseteq \mathcal{C}$

Intersection : $\forall (id_m)_{1 \leq m \leq k+1} \in \mathcal{P}^{k+1}, \forall (\tau_m)_{1 \leq m \leq k+1} \in \mathbb{N}^{k+1}$:

$$\exists (i,j) \in [\![1; k+1]\!]^2 : (i \neq j) \wedge (qr_{id_i}^{\tau_i} \cap qr_{id_j}^{\tau_j} \neq \emptyset)$$

Auto-inclusion : $\forall \tau, \forall i \in \mathcal{P} : i \in qr_i^{\tau}$

Autorité partielle : $\exists \ell \in \mathcal{C} : \forall i \in \mathcal{C}$:

$$(\forall \tau : \exists \tau_i, \tau_\ell \in [\![\tau; \infty[\![: qr_i^{\tau_i} \cap qr_\ell^{\tau_\ell} \neq \emptyset) \implies (\exists \tau : \forall \tau' > \tau : ld_i^{\tau'} = \ell)$$

La propriété d'auto-inclusion est ajoutée pour simplifier les raisonnements. On remarquera que lorsque l'on dispose de quorums qui vérifient les propriétés de vivacité et d'intersection, on peut aisément construire des quorums vérifiant de surcroît l'auto-inclusion (en prenant $qr'_i = qr_i \cup \{i\}$) tout en préservant les deux premières propriétés.

La propriété d'intersection assure que parmi les quorums fournis dans tout le système au cours d'une exécution complète, il n'y en a pas $k+1$ qui soient deux à deux disjoints. La propriété de vivacité garantit qu'à partir d'un certain temps, les quorums de processus corrects ne contiennent que des identités de processus corrects.

La propriété d'autorité partielle peut être mieux comprise en considérant la contraposée de l'implication : il y a un processus correct p_ℓ que tout processus correct p_i doit inéluctablement et pour toujours considérer comme son leader *sauf si, après un certain temps, tout quorum de p_i est disjoint de tout quorum de p_ℓ*.

5.1.2 $\Pi\Sigma_{x,y}$

On peut considérer $\Pi\Sigma_{x,y}$ comme une mise en parallèle de y exemplaires de $\Pi\Sigma_x$ parmi lesquels un seul doit vérifier la propriété d'autorité partielle.

Formellement, $\Pi\Sigma_{x,y}$ fournit aux processus un tableau de y couples de variables. Le premier élément du j^e couple sur le processus p_i sera noté $FD_i[j].qr$ et contiendra un ensemble d'identités de processus. Le second élément de ce couple sera noté $FD_i[j].ld$ et contiendra une identité de processus. On notera $FD[j]$ l'objet distribué qui à tout processus p_i fournit le couple $FD_i[j]$.

$\Pi\Sigma_{x,y}$ doit satisfaire les propriétés suivantes :

Sûreté : $\forall j \in [\![1; y]\!]$: $FD[j]$ vérifie les propriétés de vivacité, d'intersection et d'auto-inclusion de $\Pi\Sigma_x$.

Vivacité : $\exists j \in [\![1; y]\!] : FD[j]$ vérifie la propriété d'autorité partielle de $\Pi\Sigma_x$.

5.2 Comparaison avec la famille $\langle \Sigma_x \times \overline{\Omega}_y \rangle$

Dans cette section, nous nous attacherons à comparer les détecteurs de la famille $\Pi\Sigma_{x,y}$, que nous venons de définir, et ceux de la famille $\langle \Sigma_x \times \overline{\Omega}_y \rangle$.

```
(01) repeat forever rel_broadcast LEADER(leaders_i) end repeat.

(02) when LEADER(ld) is delivered :
(03)    for each j ∉ ld do susp_nb_i[j] ← susp_nb_i[j] + 1 end for ;
(04)    let j_1, j_2, ..., j_n be a permutation of {1, ..., n} such that
        (susp_nb_i[j_1], j_1) < (susp_nb_i[j_2], j_2) < ··· < (susp_nb_i[j_n], j_n) ;
(05)    for each m ∈ [[1; y]] do FD_i[m].ld ← j_x end for.

(06) repeat forever
(07)    for each m ∈ [[1; y]] do FD_i[m].qr ← qr_i ∪ {i} end for
(08) end repeat.
```

Algorithme 7: $\Pi\Sigma_{x,y}$ dans $\mathcal{AMP}[\langle\Sigma_x, \overline{\Omega}_y\rangle]$ (code pour p_i)

5.2.1 $\Pi\Sigma_{x,y} \preceq \langle\Sigma_x, \overline{\Omega}_y\rangle$

Théorème 6 *L'algorithme 7 implémente $\Pi\Sigma_{x,y}$ dans $\mathcal{AMP}[\langle\Sigma_x, \overline{\Omega}_y\rangle]$.*

Preuve Remarquons tout d'abord que la boucle des lignes 06 à 08 assurent que tous les objets distribués $FD[m], m \in [\![1; y]\!]$ vérifient les propriétés de vivacité, d'intersection et d'auto-inclusion de $\Pi\Sigma_x$. En conséquence, la sûreté de $\Pi\Sigma_{x,y}$ est garantie.

À la ligne 01, l'algorithme utilise un mécanisme de diffusion fiable qui garantit que, si un message est délivré par un processus (correct ou non), alors tous les processus correct délivreront ce même message. Ce mécanisme est implémentable dans $\mathcal{AMP}$ sans hypothèses supplémentaires.

Soit Π l'ensemble des identités de processus qui, à partir d'un certain temps, apparaissent dans tous les ensembles $leaders$ lus par des processus corrects. Selon la propriété d'autorité faible inéluctable de $\overline{\Omega}_y$, $\Pi \cap \mathcal{C} \neq \emptyset$ et, selon la propriété de validité, $|\Pi| \leq k$.

Il y a un moment à partir duquel les processus de Π apparaissent dans toutes les variables $leaders$ transportées par les messages LEADER. Par conséquent, pour chaque $p_j, j \in \Pi$, il n'y a qu'un nombre fini de messages LEADER portant une variable $leaders$ qui ne contient pas j qui circulent au cours de l'exécution. Conformément à la ligne 03, sur tous les processus corrects p_i, les compteurs $susp\_nb_i[j]$ n'augmenteront donc plus après un certain temps. De plus, grâce au mécanisme de diffusion fiable, les processus corrects reçoivent tous le même ensemble de messages LEADER, en conséquence :

$$\exists\tau : \forall j \in \Pi : \exists N_j \in \mathbb{N} : \forall\tau' \geq \tau, \forall i \in \mathcal{C} : susp\_nb_i^{\tau'}[j] = N_j$$

Au contraire, tout processus $p_j, j \in \mathcal{P} \setminus \Pi$ est infiniment souvent absent des ensembles $leaders$ lus par les processus corrects. Par conséquent, ces derniers envoient une infinité de messages portant des variables $leaders$ ne contenant pas j. Donc pour tout processus p_i correct, les compteurs $susp\_nb_i[j]$ croîtront indéfiniment.

Compte-tenu des lignes 04 et 05, il existe un moment à partir duquel, sur tout processus correct p_i et pour toujours, les variables $FD_i[m].ld$ pour $1 \leq m \leq |\Pi|$ contiendront les éléments $j \in \Pi$ triés selon l'ordre croissant sur les valeurs N_j associées.

Comme il existe $\ell \in \Pi \cap \mathcal{C}$, il existe $m_0 \in [\![1; |\Pi|]\!]$ tel que $\exists\tau : \forall\tau' \geq \tau, \forall i \in \mathcal{C} : FD_i^{\tau'}[m_0] = \ell$. Ainsi, à partir d'un certain temps, $FD_i[m_0].ld = \ell$ sur tout processus correct p_i et pour toujours. Par conséquent, la partie droite de l'implication dans la propriété d'autorité

partielle est vérifiée par l'objet $FD[m_0]$. La propriété elle-même est donc également vérifiée, ce qui prouve la propriété de vivacité de $\Pi\Sigma_{x,y}$.

$\square_{Théorème\ 6}$

5.2.2 $\Pi\Sigma_{1,y} \simeq \langle \Sigma_1, \overline{\Omega}_y \rangle$

Dans le cas où $x = 1$, $\langle \Sigma_x, \overline{\Omega}_y \rangle$ correspond au détecteur minimal permettant de simuler une mémoire partagée et de résoudre le k-accord dans le système simulé.

Théorème 7 $\Pi\Sigma_{1,y} \simeq \langle \Sigma_1, \overline{\Omega}_y \rangle$

Preuve Le Lemme 6 nous donne déjà $\Pi\Sigma_{1,y} \preceq \langle \Sigma_1, \overline{\Omega}_y \rangle$, il nous faut donc montrer $\Pi\Sigma_{1,y} \succeq \langle \Sigma_1, \overline{\Omega}_y \rangle$. Pour cela on considère que, sur tout processus p_i, on simule la sortie qr_i de Σ_1 en répétant $qr_i \leftarrow LD_i[1].qr$. Σ_1 hérite ainsi des propriétés de vivacité et d'intersection de $LD[1]$.

On suppose également que l'on simule la sortie de $\overline{\Omega}_y$, $leaders_i$, en l'affectant régulièrement avec n'importe quel ensemble de y identités de processus contenant $\bigcup_{j \in [\![1;y]\!]} LD[j].ld$ (la propriété de validité de Ω_y est ainsi vérifiée). La propriété de vivacité de $\Pi\Sigma_{1,y}$ nous assure qu'il existe $j_0 \in [\![1; y]\!]$ tel que $LD[j_0]$ vérifie la propriété d'autorité partielle de $\Pi\Sigma_1$. La propriété de sûreté de $\Pi\Sigma_{1,y}$ garantit que $LD[j_0]$ vérifie également la propriété d'intersection de $\Pi\Sigma_1$. Par conséquent, tous les quorums fournis par $LD[j_0]$ s'intersectent deux à deux. La partie gauche de l'implication dans la propriété d'autorité partielle est donc toujours vraie pour toute paire de processus correct (p_i, p_ℓ). Il en découle que pour ℓ tel que défini dans cette propriété, la partie de droite de l'implication est vérifiée par tous les processus corrects p_i. Tous les processus corrects p_i finissent donc par avoir pour toujours $LD_i[j_0].ld = \ell$, ainsi inéluctablement, ils vérifient aussi à tout moment $\ell \in leaders_i$. La propriété d'autorité faible inéluctable de Ω_y est donc vérifiée.

$\square_{Théorème\ 7}$

5.2.3 $\Pi\Sigma_{x,n-1} \simeq \Sigma_x$

Théorème 8 $\Pi\Sigma_{x,n-1} \simeq \Sigma_x$

Preuve Le théorème 6 nous donne $\Pi\Sigma_{x,n-1} \preceq \langle \Sigma_x, \overline{\Omega}_n - 1 \rangle$, il reste donc à montrer que $\Sigma_x \succeq \Pi\Sigma_{x,n-1}$. Il est montré dans [3] que $\Sigma_x \succeq \Sigma_{n-1} \succeq \Omega_{n-1}$. De plus, de par leur définition, $\Omega_{n-1} \succeq \overline{\Omega}_{n-1}$. Ainsi, $\Sigma_x \succeq \langle \Sigma_x, \overline{\Omega}_n - 1 \rangle$ et donc $\Sigma_x \succeq \Pi\Sigma_{x,n-1}$.

$\square_{Théorème\ 8}$

5.2.4 $\Pi\Sigma_{x,y} \not\succeq \langle \Sigma_x, \overline{\Omega}_y \rangle$ pour $1 \leq y < x < n$

Lemme 13 $\Pi\Sigma_{y+1,1} \not\succeq \overline{\Omega}_y$.

Preuve Supposons que l'on dispose d'un algorithme A_y qui implémente $\overline{\Omega}_y$ dans le système $\mathcal{AMP}[\Pi\Sigma_{y+1,1}]$. On considère l'ensemble E des exécutions infinies de A_y dans laquelle les processus $p_i, i > y + 1$ sont initialement crashés et $\forall \tau, \forall i \in [\![1; y + 1]\!] : qr_i^\tau = \{i\} \wedge ld_i^\tau = i$.

Nous montrons que dans une de ces exécutions, il est impossible que A_y fournisse une sortie correcte pour $\overline{\Omega}_y$.

Pour cela nous allons montrer par récurrence l'assertion $a(m)$ suivante : si dans une exécution e de E, à un instant τ, seuls les processus $p_i, i \in [\![1;m]\!]$ sont vivants, alors il existe une exécution $e' \in E$ semblable à e jusqu'à τ telle que, dans e', $\exists \tau' \geq \tau, \exists i \in [\![1;m]\!] : leaders_i^{\tau'} \supseteq [\![1;m]\!]$.

Initialisation. Dans une exécution e de E dans laquelle, à τ, tous les processus sauf p_1 sont crashés, la propriété d'autorité faible inéluctable de $\overline{\Omega}_y$ implique qu'à partir d'un certain temps on ait toujours $leaders_1 \ni 1$. $a(1)$ est donc vérifiée.

Hérédité. Supposons que $a(m)$ soit vraie pour $1 \leq m < y$. Soit $e \in E$ une exécution de A_y dans lequelles, à τ, seuls les processus $p_i \in [\![1;m+1]\!]$ sont vivants. Soit e' une exécution semblable à e jusqu'à τ mais dans laquelle p_1 crashe après τ. Selon l'hypothèse de récurrence (à un renommage des processus près), on dispose de ${e'}_1$ semblable à e' jusqu'à τ dans laquelle il existe $\tau_1 \geq \tau$ et $i_1 \in [\![2;m+1]\!]$ tels que $leaders_{i_1}^{\tau_1} \supseteq [\![2;m+1]\!]$.

Soit $e_1 \in E$ une exécution semblable à e dans laquelle les messages envoyés par p_1 après τ sont retardés jusqu'à τ_1. Les processus $p_i, i \in [\![2;m+1]\!]$ ne peuvent distinguer ${e'}_1$ de e_1 avant τ_1. Par conséquent, dans e_1 à τ_1 on a $leaders_{i_1}^{\tau_1} \supseteq [\![2;m+1]\!]$.

On peut maintenant définir de la même façon e_2 comme une exécution semblable à e_1 jusqu'à τ_1 dans laquelle il existe τ_2 et i_2 tels que $leaders_{i_2}^{\tau_2} \supseteq [\![1;m+1]\!] \setminus \{2\}$.

On peut ensuite répéter cette construction un nombre infini de fois en retardant les messages de chaque processus à tour de rôle. Si jamais aucun processus ne vérifie $leaders_i \supseteq [\![1;m+1]\!]$, alors dans l'exécution obtenue, tout processus disparait infiniment souvent d'une des variables $leaders$, ce qui contredit la propriété d'autorité faible inéluctable de $\overline{\Omega}_y$. $a(m+1)$ est donc vérifiée.

Par récurrence, la propriété $a(m)$ est vérifiée pour tout $1 \leq m \leq y$. Remarquons que la preuve de l'héredité n'utilise pas le fait que $m < y$. Par conséquent, $a(y+1)$ est vérifiée. Cependant, le fait d'avoir, sur un processus p_i, à un instant τ, $leaders_i^{\tau} \supseteq [\![1;y+1]\!]$ contredit la propriété de validité de $\overline{\Omega}_k$. A_y ne peut donc pas exister.

$\square_{Lemme\ 13}$

Théorème 9 *Pour tout triplet* (x,y,n) *tel que* $1 \leq y < x < n$, $\Pi\Sigma_{x,y} \prec \langle \Sigma_x, \overline{\Omega}_y \rangle$.

Preuve Remarquons tout d'abord que le théorème 6 nous donne $\Pi\Sigma_{x,y} \preceq \langle \Sigma_x, \overline{\Omega}_y \rangle$. Il reste donc à montrer que $\Pi\Sigma_{x,y} \not\succeq \langle \Sigma_x, \overline{\Omega}_y \rangle$.

On obtient $\Pi\Sigma_{y+1,1} \succeq \Pi\Sigma_{y+1,y}$ en considérant que l'on recopie la sortie de $\Pi\Sigma_{y+1,1}$ dans tous les couples de variables du tableau que fournit $\Pi\Sigma_{y+1,y}$.

Par ailleurs, la propriété d'intersection de $\Pi\Sigma_x$ est la seule dépendant de x et, pour tout x, celle de $\Pi\Sigma_{x-1}$ implique celle de $\Pi\Sigma_x$. On en déduit que, pour tout $x > y$, $\Pi\Sigma_{x,y} \preceq \Pi\Sigma_{y+1,y}$.

Au bilan, on obtient, pour $x > y$, $\Pi\Sigma_{y+1,1} \succeq \Pi\Sigma_{x,y}$. Comme le Lemme 13 prouve que $\Pi\Sigma_{y+1,1} \not\succeq \overline{\Omega}_y$, on en déduit que $\Pi\Sigma_{x,y} \not\succeq \overline{\Omega}_y$, et donc que $\Pi\Sigma_{x,y} \not\succeq \langle \Sigma_x, \overline{\Omega}_y \rangle$.

$\square_{Théorème\ 9}$

5.3 $\Pi\Sigma_{x,y}$ est suffisant pour le xy-accord

Après avoir montré que $\Pi\Sigma_{x,y}$ fournissait en général moins d'information sur les défaillances que $\langle \Sigma_x, \overline{\Omega}_y \rangle$, nous avons modifié légèrement l'algorithme utilisé dans [5] afin de l'adapter à notre détecteur de fautes. Nous montrons ici que nous pouvons alors résoudre le xy-accord dans $\mathcal{AMP}[\Pi\Sigma_{x,y}]$.

5.3.1 Définition de l'abstraction Alpha_x

Cette abstraction a été introduite dans [14] et généralisée au x-accord dans [23, 5]. Elle vise à capturer les propriétés de validité et d'accord du x-accord dans un objet partagé que nous noterons Alpha_x. Cet objet fournit aux processus une opération $\mathsf{propose}(r, v)$. Le premier paramètre, r, est un numéro de ronde unique : des processus distincts utilisent toujours des numéros de ronde distincts et, sur un processus donné, les appels successifs se font avec des numéros de ronde strictement croissants. Le second paramètre, v, est la valeur que le processus souhaite proposer, elle est supposée différente d'une valeur spéciale que nous noterons $\perp$. Un appel à $\mathsf{propose}(r, v)$ renvoie une valeur qui peut être $\perp$.

L'abstraction Alpha_x doit remplir ces propriétés :

Terminaison : Toute invocation de $\mathsf{propose}(-,-)$ retourne une valeur (qui peut être $\perp$) en un temps fini.

Validité : Si une invocation de $\mathsf{propose}(r,-)$ retourne une valeur $w \neq \perp$, alors l'opération $\mathsf{propose}(r', w)$ a été invoquée avec $r' \leq r$.

Sûreté : L'ensemble des valeurs différentes de $\perp$ retournées, dans tout le système, par les invocations à $\mathsf{propose}(-,-)$ est de cardinal inférieur ou égal à x.

Obligation : Soit p_ℓ un processus correct et notons $\overline{Q}(\ell,\tau) = \{i \mid \forall \tau_i, \tau_\ell \geq \tau : qr_i^{\tau_i} \cap qr_\ell^{\tau_\ell} = \emptyset\}$. Si, après un instant τ, (a) seul p_ℓ et les processus de $\overline{Q}(\ell,\tau)$ invoquent $\mathsf{propose}(-,-)$ et (b) p_ℓ invoque $\mathsf{propose}(-,-)$ infiniment souvent, alors une des invocations exécutées par p_ℓ renvoie une valeur différente de $\perp$.

Cet objet se comporte comme une mémoire dans laquelle on peut écrire au plus x valeurs et qui renvoie une des valeurs stockées ou, en cas de concurrence pour accèder à l'objet, une valeur spéciale $\perp$ pour annuler l'opération.

La propriété d'obligation que nous utilisons ici est plus forte que celle de [5], elle suppose cependant que le système sous-jacent est $\mathcal{AMP}[\Sigma_x]$. Dans [5], la propriété d'obligation garantit qu'un appel I à $\mathsf{propose}(r,-)$ renvoie une valeur non-$\perp$ lorsque tous les appels à $\mathsf{propose}(r',-)$ qui commencent avant que I ne termine portent un numéro de ronde $r' < r$. Notre propriété n'autorise pas qu'un processus p_ℓ exécute une infinité d'invocations qui retournent $\perp$ si les seuls processus qui effectuent des appels concurrents ont des quorums disjoints de ceux de p_ℓ.

5.3.2 Implémentation de Alpha_x dans $\mathcal{AMP}[\Pi\Sigma_x]$

Nous présentons ici un algorithme qui est une adaptation de celui proposé dans [5]. Il est prouvé dans cet article que l'algorithme 8 implémente, dans $\mathcal{AMP}[\Sigma_k]$, l'abstraction Alpha_x telle que définie plus haut mais avec une propriété d'obligation différente :

init $lre_i \leftarrow 0$; $est_i \leftarrow \perp$; $pos_i \leftarrow 0$.

operation propose(r, v_i) :
(01) broadcast REQ_R(r) ;
(02) **repeat** $Q_i \leftarrow qr_i$
(03) **until** ($\forall j \in Q_i$: RSP_R($r, \langle lre_j, pos_j, est_j\rangle$) received from p_j) **end repeat** ;
(04) **let** $rcv_i = \{ \langle lre_j, pos_j, est_j\rangle$: RSP_R($r, \langle lre_j, pos_j, est_j\rangle$) received $\}$;
(05) **if** ($\exists lre : \langle lre, -, -\rangle \in rcv_i : lre > lre_i$) **then** return$(\perp)$ **end if** ;
(06) $pos_i \leftarrow \max\{pos \mid \langle r, pos, v\rangle \in rcv_i\}$; $est_i \leftarrow \max\{v \mid \langle r, pos_i, v\rangle \in rcv_i\}$;
(07) **if** $(est_i = \perp)$ **then** $est_i \leftarrow v_i$ **end if** ;
(08) **while**$(pos_i < 2^r)$ **do**
(09) $pos_i \leftarrow pos_i + 1$; $pst_i \leftarrow pos_i$; % this line is executed atomically %
(10) broadcast REQ_W(r, pst_i, est_i) ;
(11) **repeat** $Q_i \leftarrow qr_i$
(12) **until** ($\forall j \in Q_i$: RSP_W(r, pst_i, $\langle lre_j, pos_j, est_j\rangle$) received from p_j) **end repeat** ;
(13) **let** $rcv_i = \{\langle lre_j, pos_j, est_j\rangle$: RSP_W(r, pst_i, $\langle lre_j, pos_j, est_j\rangle$) received$\}$;
(14) **if** ($\exists lre : \langle lre, -, -\rangle \in rcv_i : lre > r$) **then** return$(\perp)$ **end if** ;
(15) $pos_i \leftarrow \max\{pos \mid \langle r, pos, v\rangle \in rcv_i\}$; $est_i \leftarrow \max\{v \mid \langle r, pos_i, v\rangle \in rcv_i\}$
(16) **end while** ;
(17) return(est_i).

when REQ_R(rd) **received from** p_j :
(18) **if** $rd > lre_i$ **then** $pos_i \leftarrow$ g$(pos_i, rd - lre_i)$; $lre_i \leftarrow rd$ **end if** ;
(19) send RSP_R$(rd, \langle lre_i, pos_i, est_i\rangle)$ to p_j.

when REQ_W(rd, pos, est) **received from** p_j :
(20) **if** $rd \geq lre_i$ **then** $pos_i \leftarrow$ g$(pos_i, rd - lre_i)$; $lre_i \leftarrow rd$
(21) **case** $pos_j > pos_i$ **then** $est_i \leftarrow est$; $pos_i \leftarrow pos$
(22) $pos_j = pos_i$ **then** $est_i \leftarrow \max\{v_i, est\}$
(23) $pos_j < pos_i$ **then** nop
(24) **end case**
(25) **end if** ;
(26) send RSP_W$(rd, pos, \langle lre_i, pos_i, est_i\rangle)$ to p_j.

Algorithme 8: Alpha$_x$ dans $\mathcal{AMP}[\Sigma_x]$ (implémentation de [5])

Obligation : Soit I une invocation de propose$(r, -)$ effectuée par un processus correct. Si toutes les invocations de propose$(r', -)$ qui débutent avant que I ne termine portent des numéros de ronde $r' < r$, alors I retourne une valeur différente de $\perp$.

Nous nous appuierons sur leur preuve pour montrer que l'algorithme 9 implémente Alpha$_x$ selon notre définition dans $\mathcal{AMP}[\Sigma_x]$. La fonction g utilisée dans les deux algorithmes est définie comme suit : $g(\rho, \delta) = 2^\delta(\rho - 1) + 1$.

Notons tout d'abord que nous pouvons considérer, pour simplifier la preuve, que Σ_x fournit des quorums qui vérifient l'auto-inclusion.

Théorème 10 *L'algorithme 9 implémente* Alpha$_x$ *dans* $\mathcal{AMP}[\Sigma_x]$.

Preuve Remarquons pour commencer que les seules modifications apportée par l'algorithme 9 à l'algorithme 8 apparaissent sur les lignes préfixées d'un N. Ces modifications se situent dans les phases d'envoi et de réception des lignes 01 à 04 et 10 à 13. Seuls certains messages sont ignorés ou ne sont pas envoyés et ce ne sont pas ceux qui permettent la fin de la phase de réception. Dans l'algorithme 8 ces messages pourraient être retardés jusqu'à ce qu'un ensemble

quelconque d'invocations de propose$(-,-)$ aient terminé. Par conséquent, les propriétés de validité et d'accord d'Alpha$_x$ sont toujours vérifiées par l'algorithme 9.

Terminaison. Remarquons tout d'abord qu'un processus sortira inéluctablement de la boucle **while** des lignes 08 à 16 s'il ne reste pas bloqué dans la boucle **repeat** des lignes 11 et 12. Supposons qu'un processus correct p_i reste bloqué pour toujours dans une des boucles **repeat** des lignes 02 et 03 ou 11 et 12.

Selon la propriété de vivacité de Σ_x, il existe un moment τ à partir duquel on a toujours $qr_i \subseteq \mathcal{C}$. Comme le nombre de processus est fini, il y a un instant $\tau' \geq \tau$ après lequel p_i a envoyé des messages REQ_X à tous les processus qui apparaissent dans ses quorums après τ'. Chaque processus correct répond par un message RESP_X à tout message REQ_X qu'il reçoit. En conséquence, p_i finit par recevoir un message RESP_X de tous les processus qui apparaissent dans ses quorums après τ'. Mais alors la condition de sortie de la boucle **repeat** est satifaite ce qui contredit le fait que p_i y reste bloqué.

Obligation. Soient p_ℓ, τ et $\overline{Q}(\ell,\tau)$ tels que définis dans la propriété d'obligation de Alpha$_x$. Notons alors $\Pi(\ell,\tau) = \bigcup_{\tau' \geq \tau} qr_\ell^{\tau'}$ et $\overline{\Pi}(\ell,\tau) = \bigcup_{\substack{\tau' \geq \tau \\ i \in \overline{Q}(\ell,\tau)}} qr_i^{\tau'}$ Il vient de la définition de $\overline{Q}(\ell,\tau)$ que $\Pi(\ell,\tau) \cap \overline{\Pi}(\ell,\tau) = \emptyset$.

Soit $\tau_0 \geq \tau$ un instant après lequel toutes les invocations de propose$(-,-)$ commencées avant τ ont terminé et les messages qu'elles ont générés ont tous été reçus et traités.

Dans notre algorithme, les processus n'envoient de messages qu'aux processus qui apparaissent dans leurs quorums. Par conséquent, après τ_0, p_ℓ n'envoie plus de requêtes qu'aux processus de $\Pi(\ell,\tau)$ et les processus de $\overline{Q}(\ell,\tau)$ n'envoient plus de requêtes qu'aux processus de $\overline{\Pi}(\ell,\tau)$. Les seuls messages qui transitent alors sont donc échangés entre p_ℓ et les processus de $\Pi(\ell,\tau)$ ou entre les processus de $\overline{Q}(\ell,\tau)$ et ceux de $\overline{\Pi}(\ell,\tau)$. Il apparait alors que les processus de $\Pi(\ell,\tau)$ ne peuvent pas distinguer cette exécution e d'une autre e' où les processus de $\overline{\Pi}(\ell,\tau)$ auraient crashé après τ_0.

Comme après τ_0, p_ℓ invoque infiniment souvent propose$(-,-)$ avec des numéros de ronde strictement croissants et comme les processus de $\Pi(\ell,\tau)$ ne reçoivent plus de requêtes de processus autres que p_ℓ, p_ℓ finit par exécuter $I =$propose$(r,-)$ avec un numéro de ronde r strictement plus grand que ceux vus par les processus de $\Pi(\ell,\tau)$ avant τ_0.

Dans cette situation, dans e', la propriété d'obligation de [5] est satisfaite et I renvoie une valeur différente de $\perp$. Selon l'argument d'indistinguabilité, les processus de $\Pi(\ell,\tau)$ se comportent dans e comme dans e' et I renvoie donc également une valeur différente de $\perp$ dans e.

$\square_{Théorème\ 10}$

5.3.3 Résolution du xy-accord dans $\mathcal{AMP}[\Pi\Sigma_{x,y}]$

x-accord dans $\mathcal{AMP}[\Pi\Sigma_x]$

Nous supposons ici que l'on a construit l'objet Alpha$_x$, ce qui est possible puisque $\Pi\Sigma_x \succeq \Sigma_x$ et que l'on a donné une implémentation de Alpha$_x$ dans $\mathcal{AMP}[\Sigma_k]$.

Lemme 14 *L'algorithme 10 résoud le x-accord dans $\mathcal{AMP}[\Pi\Sigma_x]$.*

Preuve Remarquons tout d'abord que les lignes 01, 04 et 05 assurent que les appels à Alpha_x se font dans les conditions requises : des numéros de ronde croissants pour les appels consécutifs effectués par un même processus et des numéros de rondes différents pour tous les appels en provenance de processus différents.

Grâce aux propriétés de sûreté et de validité de Alpha_x, comme les seules valeurs décidées proviennent de la sortie de $\mathsf{Alpha}_x.\mathsf{propose}(-,-)$ et que les seules valeurs proposées à Alpha_x sont les valeurs proposées par les processus, les propriétés d'accord et de validité du x-accord sont vérifiées. De plus, la propriété de terminaison d'Alpha_x nous garantit qu'aucun processus ne restera bloqué indéfiniment pendant une invocation d'Alpha_x. Enfin, le mécanisme de diffusion fiable utilisé à la ligne 08 assure que si un processus décide, alors tous les processus corrects finissent par décider.

Ainsi, il ne nous reste qu'à montrer que l'une des invocations de $\mathsf{Alpha}_x.\mathsf{propose}(-,-)$ renvoie une valeur différente de $\perp$. Supposons que ça ne soit jamais le cas. Soit ℓ une identité de processus correct telle que celle définie dans la propriété d'autorité partielle de $\Pi\Sigma_x$. Soit R_ℓ l'ensemble des processus p_i tels que $i \neq l$ et $ld_i = i$ est vrai infiniment souvent. La contraposée de la propriété d'autorité partielle nous assure qu'il existe un instant τ_{R_ℓ} tel que tous les processus $p_i, i \in R_\ell$ vérifient $\forall \tau_i, \tau_l \geq \tau_{R_\ell} : qr_i^{\tau_i} \cap qr_\ell^{\tau_\ell} = \emptyset$. Ainsi, $R_\ell \subseteq \overline{Q}(\ell, \tau_{R_\ell})$ avec $\overline{Q}$ défini comme dans la propriété d'obligation de Alpha_x.

Conformément à la condition de la ligne 03, il existe un instant τ_a après lequel seuls les processus de $R_\ell \cup \{\ell\}$ appellent $\mathsf{Alpha}_x.\mathsf{propose}(-,-)$. Soit $\tau_b = \max\{\tau_a, \tau_{R_\ell}\}$, par définition de $\overline{Q}$, on a $R_\ell \subseteq \overline{Q}(\ell, \tau_{R_\ell}) \subseteq \overline{Q}(\ell, \tau_b)$. Grâce aux propriété d'autorité partielle et d'auto-inclusion de $\Pi\Sigma_x$, il existe un instant après lequel on a toujours $ld_\ell = \ell$, donc p_ℓ appelle infiniment souvent $\mathsf{Alpha}_x.\mathsf{propose}(-,-)$, car, par hypothèse, aucun processus ne décide.

p_ℓ et τ_b ainsi définis vérifient donc les conditions de la propriété d'obligation d'Alpha_x. Un des appels effectués par p_ℓ à $\mathsf{Alpha}_x.\mathsf{propose}(-,-)$ renvoie donc une valeur différente de $\perp$. Nous avons atteint une contradiction, par conséquent un processus finit par décider et tous finissent donc par décider, ce qui prouve la propriété de terminaison du x-accord.

$\square_{Lemme\ 14}$

xy-accord dans $\mathcal{AMP}[\Pi\Sigma_{x,y}]$

Théorème 11 *En exécutant en parallèle y instances de l'algorithme 10, on résoud le xy-accord dans $\mathcal{AMP}[\Pi\Sigma_{x,y}]$.*

Preuve Pour résoudre le xy-accord dans $\mathcal{AMP}[\Pi\Sigma_{x,y}]$, chaque processus exécute y instances de l'algorithme 10 en parallèle et décide la première valeur décidée dans une de ces instances. La j^e de ces instances utilise le leader fourni par $FD[j].ld$ et un objet Alpha_x implémenté grâce aux quorums fournis par $FD[j].qr$.

La propriété de sûreté de $\Pi\Sigma_{x,y}$ assure que tous les objets $FD[j], j \in [\![1; y]\!]$ satisfont les propriétés d'intersection d'auto-inclusion et de vivacité de $\Pi\Sigma_x$ (qui sont celles de Σ_x). On peut donc bien implémenter Alpha_x à partir de chacun d'eux. Ainsi, pour tout $j \in [\![1; y]\!]$, la j^e instance de l'algorithme satisfait les propriétés d'accord et de validité du x-accord qui ne dépendent que d'Alpha_x (cf. preuve du Lemme 14).

Par ailleurs, la propriété de vivacité de $\Pi\Sigma_{x,y}$ nous assure qu'il existe $j_0 \in [\![1; y]\!]$ tel que $FD[j_0]$ satisfait la propriété d'autorité partielle de $\Pi\Sigma_x$. Par conséquent, la ${j_0}^e$ instance de l'algorithme vérifie la propriété de terminaison du x-accord.

Les propriétés d'accord et de validité étant vérifiées par les y instances du x-accord, au plus xy valeurs sont décidées dans tout le système et ce sont toutes des valeurs proposées. Les propriétés d'accord et de validité du xy-accord sont donc garanties. Par ailleurs, au moins une des y instances du x-accord termine, par conséquent, s'ils ne décident pas dans une autre instance, tous les processus corrects finissent par décider dans celle-ci. La propriété de terminaison du xy-accord est donc également vérifiée.

$\square_{Théorème\ 11}$

Nous disposons ainsi d'un détecteur de faute plus faible que $\langle \Sigma_x, \overline{\Omega}_y \rangle$ dans le cas général et qui permet de résoudre le k-accord dans les mêmes conditions. Cependant, remarquons que notre algorithme ne fonctionne que pour $xy \leq k$. Ainsi, même si $\Pi\Sigma_{n-1,n-1} \simeq \Sigma_{n-1}$ est le détecteur de faute minimal pour résoudre le $n-1$-accord, notre algorithme ne permet pas d'implémenter le $n-1$-accord dans $\mathcal{AMP}[\Pi\Sigma_{n-1,n-1}]$.

5.4 Ω_k n'est pas nécessaire pour le k-accord

Les détecteurs de fautes des trois familles $\square\Pi_k$, $\mathcal{L}_k$ et $\mathcal{S}_{n-k+1}$ étaient tous plus forts que Ω_k, en effet, $\square\Pi_k \succeq \Pi_k \simeq \langle \Sigma_k, \Omega_k \rangle$ et nous avons montré $\mathcal{L}_k \succeq \square\Pi_k$ et $\mathcal{S}_{n-k+1} \succeq \square\Pi_k$. Nous nous sommes alors demandé si Ω_k était nécessaire pour résoudre le k-accord.

Dans cette partie, nous montrons que ce n'est pas le cas en prouvant que $\Pi\Sigma_{k,1}$, avec lequel il est possible de résoudre le k-accord, ne permet pas de simuler Ω_k.

Lemme 15 *Soit $k \in]\!]1; n-1[\![$. Ω_k ne peut pas être simulé dans $\mathcal{AMP}[\Pi\Sigma_{k,1}]$.*

Preuve Nous supposons par contradiction que nous disposons d'un algorithme A qui simule Ω_k dans $\mathcal{AMP}[\Pi\Sigma_{k,1}]$ avec $1 < k < n-1$. Nous allons alors construire une exécution de A dans laquelle on dispose d'une sortie correcte pour $\Pi\Sigma_{k,1}$, mais où il est impossible de construire une sortie de Ω_k.

Nous noterons qr_i et ld_i les sorties de $\Pi\Sigma_{k,1}$ sur le processus p_i et $leader_i$ celle de Ω_k. Dans toutes les exécutions qui suivent, les processus $p_i, i > k+2$ (s'il en existe) sont initialement crashés.

Les exécutions α. Dans toutes les exécution de cette partie, nous considérons qu'il existe $lead_\alpha \in [\![1;3]\!]$ tel qu'à tout instant τ :

- $\forall i \in [\![1;3]\!] : (qr_i^\tau = [\![1;3]\!]) \wedge (ld_i^\tau = lead_\alpha)$
- $\forall i \in [\![4;k+2]\!] : (qr_i^\tau = \{i\}) \wedge (ld_i^\tau = i)$

Soient tout d'abord $\alpha_i, i \in [\![4;k+2]\!]$ des exécutions dans lequelles tous les processus sauf p_i sont initiallement crashés. Soit $\alpha_{[\![1;3]\!]}$ une exécution dans laquelle tous les processus de $\mathcal{P} \setminus [\![1;3]\!]$ sont initiallement crashés. Dans $\alpha_i, i \in [\![4;k+2]\!]$, p_i doit, à partir d'un certain temps τ_i, toujours satisfaire $i \in leaders_i$ (car il est le seul processus correct) pour vérifier la propriété d'autorité forte inéluctable de Ω_k. Pour garantir cette même propriété, dans $\alpha_{[\![1;3]\!]}$, il doit exister un processus ℓ_α et un instant $\tau_{[\![1;3]\!]}$ après lequel les processus $p_i, i \in [\![1;3]\!]$ doivent toujours avoir $\ell_\alpha \in leaders_i$.

Soient alors $\tau_\alpha = \max\{\tau_i, i \in \{[\![1;3]\!]\} \cup [\![4;k+2]\!]\}$ et α une exécution dans laquelle tous les messages hormis ceux envoyés entre les processus $p_i, i \in [\![1;3]\!]$ sont retardés jusqu'à τ_α. Les processus $p_i, i \in [\![1;3]\!]$ ne peuvent pas distinguer α de $\alpha_{[\![1;3]\!]}$ et les processus $p_i, i \in [\![4;k+2]\!]$

ne peuvent distinguer α de α_i. En conséquence, dans α, on a $\forall i \in [\![1;3]\!] : \ell_\alpha \in leaders_i^{\tau_\alpha}$ et $\forall i \in [\![4;k+2]\!] : i \in leaders_i^{\tau_\alpha}$.

Les exécutions β. Soit $\tau_\alpha^f > \tau_\alpha$ un instant après lequel, dans α, les messages envoyés avant τ_α ont été reçus. Les exécutions de cette partie sont toutes semblables à α jusqu'à τ_α^f. Les sorties de $\Pi\Sigma_{k,1}$ dans ces exécutions sont les mêmes que dans α jusqu'à τ_α^f. Après celà, il existe $lead_\beta \in [\![1;3]\!] \setminus \{\ell_\alpha\}$ tel qu'à tout instant $\tau > \tau_\alpha^f$ on ait :

- $\forall i \in [\![1;3]\!] \setminus \{\ell_\alpha\} : (qr_i^\tau = [\![1;3]\!] \setminus \{\ell_\alpha\}) \wedge (ld_i^\tau = lead_\beta)$
- $(qr_{\ell_\alpha}^\tau = [\![1;3]\!]) \wedge (ld_{\ell_\alpha}^\tau = lead_\beta)$

Les autres sorties de $\Pi\Sigma_{k,1}$ restent les mêmes.

Soient $\beta_i, i \in [\![4;k+2]\!]$ des exécutions semblables à α jusqu'à τ_α^f dans lesquelles tous les processus sauf p_i crashent après τ_α^f. Soit également $\beta_{[\![1;3]\!]\setminus\{\ell_\alpha\}}$ une exécution, semblable à α jusqu'à τ_α^f, dans laquelle les processus $p_i, i \notin [\![1;3]\!] \setminus \{\ell_\alpha\}$ crashent après τ_α^f. Toujours pour vérifier la propriété d'autorité forte inéluctable de Ω_k, il faut que, pour tout $i \in [\![4;k+2]\!]$, il existe τ_i tel que $\tau_i > \tau_\alpha^f$ et $\forall \tau \geq \tau_i : i \in leaders_i$ dans β_i. De même, il faut qu'il existe $\tau_{[\![1;3]\!]\setminus\{\ell_\alpha\}} > \tau_\alpha^f$ et $\ell_\beta \in [\![1;3]\!] \setminus \{\ell_\alpha\}$ tels que $\forall \tau \geq \tau_{[\![1;3]\!]\setminus\{\ell_\alpha\}}, \forall i \in [\![1;3]\!] \setminus \{\ell_\alpha\} : \ell_\beta \in leaders_i^\tau$ dans β_i.

On définit alors β, une exécution semblable à α jusqu'à τ_α^f, dans laquelle, après τ_α^f, les messages, hormis ceux envoyés par les processus $p_i, i \in [\![1;3]\!] \setminus \{\ell_\alpha\}$ et à destination d'un des processus $p_j, j \in [\![1;3]\!]$, sont retardés jusqu'à $\tau_\beta = \max\{\tau_i, i \in \{[\![1;3]\!] \setminus \{\ell_\alpha\}\} \cup [\![4;k+2]\!]\}$.

Les processus $p_i, i \in [\![1;3]\!] \setminus \{\ell_\alpha\}$ ne peuvent distinguer β de $\beta_{[\![1;3]\!]\setminus\{\ell_\alpha\}}$ et les processus $p_i, i \in [\![4;k+2]\!]$ ne peuvent distinguer β de β_i. Par conséquent, dans l'exécution β on a $\forall i \in [\![1;3]\!] \setminus \{\ell_\alpha\} : \ell_\beta \in leaders_i^{\tau_\beta}$ et $\forall i \in [\![4;k+2]\!] : i \in leader_i^{\tau_\beta}$.

Les exécutions γ. On peut maintenant définir $\tau_\beta^f > \tau_\beta$ un instant dans β après que tous les messages envoyés avant τ_β ont été reçus. On construit alors une exécution γ à partir de β de la même manière que celle utilisée pour contruire β à partir de α. On obtient ainsi une exécution dans laquelle $\exists \ell_\gamma \in [\![1;3]\!] \setminus \{\ell_\beta\}, \exists \tau_\gamma : (\forall i \in [\![1;3]\!] \setminus \{\ell_\beta\}, \ell_\gamma \in leaders_i^{\tau_\gamma}) \wedge (\forall i \in [\![4;k+2]\!] : i \in leaders_i^{\tau_\gamma})$.

En répétant cette construction, on obtient une exécution infinie dans laquelle au moins $k+1$ identités de processus (toutes celles de $[\![4;k+2]\!]$ et au moins deux parmi celles de $[\![1;3]\!]$) apparaissent infiniment souvent dans les variables $leaders$. Cela contredit la propriété d'autorité forte inéluctable de Ω_k alors que la sortie de $\Pi\Sigma_{k,1}$ pour cette exécution est correcte (en particulier, la propriété d'intersection est vérifiée au long de toutes les exécutions construites). L'agorithme A ne peut donc pas exister.

$\square_{Lemme\ 15}$

Le théorème suivant est la conséquence du Lemme 15 et du fait que le k-accord peut être résolu dans $\mathcal{AMP}[\Pi\Sigma_{k,1}]$.

Théorème 12 *Soit $k \in]\!]1;n-1[\![$. Ω_k n'est pas nécessaire pour résoudre le k-accord dans $\mathcal{AMP}$.*

Corollaire 2 *Soit $k \in]\!]1;n-1[\![$. Si $2k^2 \leq n$, alors Ω_k et le détecteur de faute minimal pour le k-accord dans $\mathcal{AMP}$ sont incomparables.*

Preuve Soit X_k le détecteur de fautes minimal pour résoudre le k-accord dans $\mathcal{AMP}$ avec $1 < k < n$ et $2k^2 \leq n$. Supposons que $\Omega_k \succeq X_k$. Il en découle que l'on peut résoudre le

k-accord dans $\mathcal{AMP}[\Omega_k]$, ce qui est une contradiction puisqu'on sait [5] que c'est impossible dans $\mathcal{AMP}[\langle \Sigma_k, \Omega_k \rangle]$ lorsque $2k^2 \leq n$.

Supposons maintenant que $X_k \succeq \Omega_k$, dans ce cas on aurait $\Pi\Sigma_{k,1} \succeq X_k \succeq \Omega_k$ ce qui contredirait le Lemme 15.

$\Box_{Corollaire\ 2}$

Nous savons donc maintenant que notre recherche doit s'orienter sur des détecteurs de fautes qui ne sont pas plus fort que Ω_k. Ce résultat prouve par ailleurs que, comme nous le soupçonnions, $\Box\Pi_k$ n'est pas le détecteur minimal pour le k-accord.

Après cette étude, $\Pi\Sigma_{x,y}$ apparait comme une version affaiblie de la paire $\langle \Sigma_x, \overline{\Omega}_y \rangle$ qui permet toujours le xy-accord.

init $lre_i \leftarrow 0$; $est_i \leftarrow \perp$; $pos_i \leftarrow 0$.

operation propose(r, v_i) :
(N01) $r\_req\_set_i \leftarrow \emptyset$;
(02) **repeat** $Q_i \leftarrow qr_i$;
(N02-1) **for each** $j \in Q_i \setminus r\_req\_set_i$ **do** send REQ_R(r) to p_j **end for** ;
(N02-2) $r\_req\_set_i \leftarrow r\_req\_set_i \cup Q_i$
(03) **until** ($\forall j \in Q_i$: RSP_R(r, $\langle lre_j, pos_j, val_j\rangle$) received from p_j) **end repeat** ;
(N04) **let** $rcv_i = \{\ \langle lre_j, pos_j, est_j\rangle$: RSP_R($r, \langle lre_j, pos_j, est_j\rangle$) received from $Q_i\ \}$;
(05) **if** ($\exists lre : \langle lre, -, -\rangle \in rcv_i : lre > lre_i$) **then** return($\perp$) **end if** ;
(06) $pos_i \leftarrow \max\{pos \mid \langle r, pos, v\rangle \in rcv_i\}$; $est_i \leftarrow \max\{v \mid \langle r, pos_i, v\rangle \in rcv_i\}$;
(07) **if** ($est_i = \perp$) **then** $est_i \leftarrow v_i$ **end if** ;
(08) **while** ($pos_i < 2^r$) **do**
(09) $pos_i \leftarrow pos_i + 1$; $pst_i \leftarrow pos_i$; % this line is executed atomically %
(N10) $w\_req\_set_i \leftarrow \emptyset$;
(11) **repeat** $Q_i \leftarrow qr_i$;
(N11-1) **for each** $j \in Q_i \setminus w\_req\_set_i$ **do** send REQ_W(r, pst_i, est_i) to p_j **end for** ;
(N11-2) $w\_req\_set_i \leftarrow w\_req\_set_i \cup Q_i$
(12) **until** ($\forall j \in Q_i$: RSP_W($r, pst_i, \langle lre_j, pos_j, val_j\rangle$) received from p_j) **end repeat** ;
(N13) $rcv_i \leftarrow \{\langle lre_j, pos_j, est_j\rangle$: RSP_W(r, pst_i, $\langle lre_j, pos_j, est_j\rangle$) received from $Q_i\ \}$;
(14) **if** ($\exists lre : \langle lre, -, -\rangle \in rcv_i : lre > r$) **then** return($\perp$) **end if** ;
(15) $pos_i \leftarrow \max\{pos \mid \langle r, pos, v\rangle \in rcv_i\}$; $est_i \leftarrow \max\{v \mid \langle r, pos_i, v\rangle \in rcv_i\}$
(16) **end while** ;
(17) return(est_i).

when REQ_R(rd) **received from** p_j :
(18) **if** $rd > lre_i$ **then** $pos_i \leftarrow$ g(pos_i, $rd - lre_i$) ; $lre_i \leftarrow rd$ **end if** ;
(19) send RSP_R(rd, $\langle lre_i, pos_i, est_i\rangle$) to p_j.

when REQ_W(rd, pos, est) **received from** p_j :
(20) **if** $rd \geq lre_i$ **then** $pos_i \leftarrow$ g($pos_i, rd - lre_i$) ; $lre_i \leftarrow rd$
(21) **case** $pos_j > pos_i$ **then** $est_i \leftarrow est$; $pos_i \leftarrow pos$
(22) $pos_j = pos_i$ **then** $est_i \leftarrow \max\{v_i, est\}$
(23) $pos_j < pos_i$ **then** nop
(24) **end case**
(25) **end if** ;
(26) send RSP_W($rd, pos, \langle lre_i, pos_i, est_i\rangle$) to p_j.

Algorithme 9: Alpha$_x$ dans $\mathcal{AMP}[\Sigma_x]$

operation propose$_x$ (v_i) :
(01) $dec_i \leftarrow \perp$; $r_i \leftarrow i$;
(02) **while** ($dec_i = \perp$) **do**
(03) **if** ($ld_i = i$) **then**
(04) $dec_i \leftarrow$ Alpha$_x$.propose(r_i, v_i)
(05) $r_i \leftarrow r_i + n$
(06) **end if**
(07) **end while** ;
(08) rel_broadcast DECISION(dec_i).

when DECISION(d) **is delivered** : return d.

Algorithme 10: x-accord dans $\mathcal{AMP}[\Pi\Sigma_x]$ (code pour p_i)

Chapitre 6

Conclusion

Au cours de ce stage, les détecteurs de fautes qui avaient été proposés jusque là ont été étudiés, comparés et certaines de leurs différences ont été explicitées. Cela nous a permis de mieux cerner celles de leur propriétés qui permettaient la résolution du k-accord dans $\mathcal{AMP}$ et nous a mené à définir deux nouvelles classes de détecteurs de fautes.

Ces nouveaux détecteurs, $\Box\Pi_k$ et $\Pi\Sigma_{x,y}$, constituent une avancée vers le détecteur de fautes minimal pour le k-accord car ils affaiblissent trois des familles connues pour être suffisantes pour résoudre ce problème et ils permettent toujours cette résolution. Ainsi, parmi les familles suffisantes connues au début de ce stage, seule $\Sigma_z, k \geq n - \lfloor \frac{n}{z+1} \rfloor$ n'a pas été améliorée.

Par ailleurs, nous avons montré que Ω_k, qui était une brique de base pour plusieurs détecteurs proposés pour le k-accord, n'est pas nécessaire pour résoudre ce problème. Ceci permet de restreindre le champs des future investigations visant à déterminer le détecteur minimal.

Il reste désormais à trouver un détecteur de fautes qui, tout en permettant la résolution du k-accord, soit plus faible que $\Box\Pi_k$, que $\Pi\Sigma_{x,y}$ pour tout couple (x, y) tel que $xy \leq k$ et que Σ_z pour z maximal tel que $k \geq n - \lfloor \frac{n}{z+1} \rfloor$. Mieux encore, il s'agit de trouver le plus faible d'entre eux.

Bibliographie

[1] Biely M., Robinson P. and Schmid U., Weak Synchrony Models and Failure Detectors for Message Passing (k-)Set Agreement. *Proc. 11th Int'l Conference on Principles of Distributed Systems (OPODIS'09)*, Springer Verlag LNCS #5923, pp. 285-299, 2009.

[2] Bonnet F. and Raynal M., A Simple Proof of the Necessity of the Failure Detector Σ to Implement an Atomic Register in Asynchronous Message-passing Systems. *Information Processing Letters*, 110(4) :153-157, 2010.

[3] Bonnet F. and Raynal M., On the Road to the Weakest Failure Detector for k-Set Agreement in Message-passing Systems. To appear in *Theoretical Computer Science*, 2011.

[4] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, San Diego (CA), pp. 91-100, 1993.

[5] Bouzid Z. and Travers C., (Anti-$\Omega_k \times \Sigma_k$)-Based k-Set Agreement Algorithms. *Proc. 12th Int'l Conference on Principles of Distributed Systems (OPODIS'10)*, Springer Verlag LNCS #6490, pp. 190-205, 2010.

[6] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2) :225-267, 1996.

[7] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4) :685–722, 1996.

[8] Chaudhuri S., More *Choices* Allow More *Faults :* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105 :132-158, 1993.

[9] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Sharing is harder than agreeing. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 85-94, Toronto (Canada), 2008.

[10] Delporte-Gallet C., Fauconnier H. and Guerraoui R., Tight Failure Detection Bounds on Atomic Object Implementations. *Journal of the ACM*, 57(4) :Article 22, 2010.

[11] Delporte-Gallet C., Fauconnier H., Guerraoui R. and Tielmann A., The Weakest Failure Detector for Message Passing Set-Agreement. *Proc. 22th Int'l Symposium on Distributed Computing (DISC'08)*, Springer-Verlag LNCS #5218, pp. 109-120, 2008.

[12] Delporte-Gallet C., Fauconnier H., Guerraoui R., A Realistic Look At Failure Detectors. *DSN 2002*, pp. 345-353, 2002.

[13] Gafni E. and Kuznetzov P., The Weakest Failure Detector for Solving k-Set Agreement. *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC'09)*, ACM Press, pp. 83-91, 2009.

[14] Guerraoui R. and Raynal M., The Alpha of Indulgent Consensus. *The Computer Journal*, 50(1) :53-67, 2007.

[15] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6) :858-923,, 1999.

[16] Jayanti P. and Toueg S., Every problem has a weakest failure detector. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 75-84, 2008.

[17] Mostéfaoui A. and Raynal M., k-Set Agreement with Limited Accuracy Failure Detectors. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, 2000.

[18] Mostéfaoui A. and Raynal M., Unreliable failure detectors with limited scope accuracy and an application to consensus. *FSTTCS*, pp. 329–340, 1999.

[19] Mostéfaoui A., Raynal M. and Stainer J., Relations Linking Failure Detectors Associated with k-Set Agreement in Message-passing Systems. *Tech Report #1973*, 13 pages, IRISA, Université de Rennes (France), April 2011. Submitted to publication.

[20] Neiger G., Failure Detectors and the Wait-free Hierarchy. *14th ACM Symposium on Principles of Distributed Computing (PODC'95)*, ACM Press, pp. 100-109, Las Vegas -NV), 1995.

[21] Raynal M., K-anti-Omega. *Rump Session at 26th ACM Symposium on Principles of Distributed Computing (PODC'07)*, 2007.

[22] Raynal M., Failure Detectors to solve Asynchronous k-set Agreement : a Glimpse of Recent Results. *The Bulletin of EATCS*, 103 :75-95, 2011.

[23] Raynal M. and Travers C., In Search of the Holy Grail : Looking for the Weakest Failure Detector for Wait-free Set Agreement. *Proc. 10th Int'l Conference On Principles Of Distributed Systems (OPODIS'06)*, Springer-Verlag LNCS #4305, pp. 1-17, 2006.

[24] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible : The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5) :1449-1483, 2000.

[25] Zielinski P., Anti-Omega : the Weakest Failure Detector for Set Agreement. *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, ACM Press, pp. 55-64, Toronto (Canada), 2008.